

# GOD DOES NOT PLAY DICE... AND NEITHER SHOULD APPROXIMATION ALGORITHMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Anke Renata van Zuijlen

August 2008

© 2008 Anke Renata van Zijlen  
ALL RIGHTS RESERVED

# GOD DOES NOT PLAY DICE... AND NEITHER SHOULD APPROXIMATION ALGORITHMS

Anke Renata van Zuijlen, Ph.D.

Cornell University 2008

We consider optimization problems for which the best known approximation algorithms are randomized algorithms: these algorithms make random choices during their execution, and it has been shown that in expectation the cost of the algorithm's solution is at most a known constant factor more than optimal. We show how to give deterministic variants of these algorithms that have similar performance guarantees. In particular, we give conditions under which the Sample-Augment algorithms proposed by Gupta et al. [42] can be derandomized, thus obtaining the best known deterministic algorithms for a number of network design problems such as the connected facility location, virtual private network design and single sink buy-at-bulk problems. We also give deterministic variants of the “pivoting” algorithms proposed by Ailon et al. [4] for several ranking and clustering problems. In addition to obtaining the same performance guarantees, the analysis of our algorithms is actually simpler than that of their randomized counterparts. Finally, we take a more practical approach to one of the ranking problems considered: the rank aggregation problem. We perform an extensive evaluation of several known and new algorithms for rank aggregation on web search data. We argue that there are two important classes of algorithms for rank aggregation: positional methods and comparison sort methods. We find that hybrid algorithms, that combine a positional and comparison sort approach, work especially well on our data sets.

## **BIOGRAPHICAL SKETCH**

Anke van Zuylen was born in 1975 in Hilvarenbeek, the Netherlands. Her journey to the Ph.D. program in Operations Research at Cornell University consisted of 5 years of studying Econometrics & Operations Research at the Vrije University in Amsterdam, where she received a Master's degree in August 2000, and quite a few detours including a year of studying Japanese in Leiden, many months of traveling the world during her studies in Amsterdam, a year as a consultant at TNT Post Group in The Hague, and some time in Ithaca doing volunteer work and working as a "professional TA". She entered the Ph.D. program in Operations Research at Cornell University in January 2004, and focused on the area of Mathematical Programming with minors in Manufacturing & Applied Operations Research and Computer Science. After completing her Ph.D., Anke will be moving to Beijing, China to take a postdoctoral position at the Institute for Theoretical Computer Science at Tsinghua University.

## ACKNOWLEDGEMENTS

Cornell's department of Operations Research and Industrial Engineering is a great place to be working towards a Ph.D., and I am very thankful for the excellent education, friendship, collegiality and support that I have experienced over the past years. In particular, many thanks go out to my advisor David Williamson. Over these past four years, David has been a terrific mentor and advisor, and I am especially grateful for the fact that he was always available for advice and that he always kept my best interest in mind.

I would also like to thank my other two committee members Shane Henderson and David Shmoys. Together with Sid Resnick, David Shmoys also deserves many thanks for giving me the opportunity to work as a teaching assistant at the ORIE department many years ago, even though I was not a Ph.D. student. Without that opportunity I may have left Ithaca years ago, and would never have considered doing a Ph.D.!

I would like to thank Emmanuel Sharef for help in extracting the search results from the four search engines used to obtain one of the data sets in Chapter 4. Emmanuel also deserves credit for his magical ability to make things work, just by looking over my shoulder, which proved necessary on more than one occasion.

Then there are of course the many people who made life at the OR department fun (or when things were not going so well, they at least made them bearable), and I thank all the students I met here over the years who made the OR department a great place to study. In particular, I was lucky to have the best office mates (and office) with Jie Chen, Emmanuel Sharef, Yun Shi and Stefan Wild. Also, many thanks go to Tuncay Alparslan for always being available to hang out while he still lived in Ithaca, and to Gavin Hurley for the many bike rides and the great parties he threw with Sam Steckley. Thanks also to Tim Carnes, Sam Ehrlichman, Retsef Levi, Chandra Nagarajan and Yogi Sharma for their advice, being willing to sit through my practice talks, and generally being great colleagues.

I thank my parents and brother and sister for their support and restraint from complaining that they did not get to see me much. Finally, I thank Frans for everything – without him I would probably never have come to Cornell, and he deserves a knighthood for his help, encouragement, but most importantly the fact that he managed to put up with me through the whole process.

## TABLE OF CONTENTS

Biographical Sketch . . . . .	iii
Acknowledgements . . . . .	iv
Table of Contents . . . . .	vi
List of Tables . . . . .	viii
List of Figures . . . . .	ix
<b>1 Introduction</b>	<b>1</b>
<b>2 Deterministic Sample-Augment Algorithms for Network Design Problems</b>	<b>8</b>
2.1 Introduction . . . . .	8
2.1.1 Related Work . . . . .	10
2.1.2 Our Results . . . . .	12
2.2 Derandomization of Sample-Augment Algorithms . . . . .	16
2.2.1 Single Source Rent-or-Buy . . . . .	20
2.2.2 2-Stage Stochastic Steiner Tree with Independent Decisions . . . . .	23
2.2.3 <i>A Priori</i> Traveling Salesman with Independent Decisions . . . . .	25
2.2.4 Connected Facility Location Problems . . . . .	29
2.2.5 Virtual Private Network Design . . . . .	35
2.3 Multi-Stage Sample-Augment Algorithms . . . . .	38
2.3.1 Single-Sink Buy-at-Bulk Network Design . . . . .	39
2.4 Conclusion . . . . .	62
<b>3 Deterministic Pivoting Algorithms for Ranking and Clustering</b>	<b>64</b>
3.1 Introduction . . . . .	64
3.1.1 Related Work . . . . .	70
3.1.2 Our Results . . . . .	72
3.2 A Simple Ranking Algorithm . . . . .	73
3.2.1 Weighted Feedback Arc Set with Triangle Inequality . . . . .	78
3.2.2 Rank Aggregation . . . . .	79
3.2.3 Weighted Feedback Arc Set with Probability Constraints . . . . .	83
3.3 A Simple Clustering Algorithm . . . . .	85
3.3.1 Weighted Clustering with Triangle Inequality . . . . .	89
3.3.2 Consensus Clustering . . . . .	90
3.3.3 Correlation Clustering . . . . .	93
3.4 Constrained Problems . . . . .	95
3.4.1 Ranking and Clustering with Triangle Inequality . . . . .	96
3.4.2 Ranking and Clustering with Probability Constraints . . . . .	99
3.5 Better LP Based Algorithms . . . . .	103
3.5.1 Weighted Feedback Arc Set . . . . .	104
3.5.2 Weighted Clustering . . . . .	111
3.6 Hierarchical Clustering . . . . .	115

<b>4</b>	<b>Rank Aggregation on Web Data: Positional, Comparison Sort and Hybrid Algorithms</b>	<b>124</b>
4.1	Introduction . . . . .	124
4.1.1	Related Work . . . . .	128
4.2	Rank Aggregation Algorithms . . . . .	129
4.2.1	Positional Algorithms . . . . .	130
4.2.2	Comparison Sort Algorithms . . . . .	133
4.2.3	Existing Hybrid Algorithms . . . . .	135
4.2.4	New Hybrids: Positional plus Comparison Sort Algorithms . . .	137
4.2.5	Other Approaches . . . . .	138
4.3	Lower Bounds on Guarantees . . . . .	140
4.3.1	Positional Methods . . . . .	140
4.3.2	Comparison Sort Methods . . . . .	143
4.3.3	Hybrid Algorithms . . . . .	145
4.4	Evaluation . . . . .	150
4.4.1	Description of Data Sets . . . . .	150
4.4.2	Results . . . . .	151
4.4.3	Randomized versus Deterministic QuickSort . . . . .	155
4.5	Conclusion . . . . .	158
<b>A</b>	<b>Tables with Results from Chapter 4</b>	<b>160</b>
	<b>Bibliography</b>	<b>162</b>



## LIST OF TABLES

2.1	Summary of Best Known Approximation Guarantees for the Problems Considered in Chapter 2. . . . .	15
A.1	Running Time and Performance of the Rank Aggregation Algorithms on the Web Communities Data Set . . . . .	160
A.2	Running Time and Performance of the Rank Aggregation Algorithms on the Web Search Data Set . . . . .	161

## LIST OF FIGURES

2.1	Sample-Augment Algorithm . . . . .	16
2.2	Sample-Augment Algorithm for Single Source Rent-or-Buy . . . . .	21
2.3	Sample-Augment Algorithm for the <i>A Priori</i> Traveling Salesman Problem . . . . .	27
2.4	Sample-Augment Algorithm for Connected Facility Location . . . . .	30
2.5	Sample-Augment Algorithm for Virtual Private Network Design . . . . .	36
2.6	The Redistribute Subroutine . . . . .	41
2.7	Sample-Augment Algorithm for Single-Sink Buy-at-Bulk . . . . .	43
2.8	The $k$ -th Stage of the Sample-Augment Algorithm for Single-Sink Buy-at-Bulk . . . . .	44
2.9	Sample-Augment Algorithm for Single-Sink Buy-at-Bulk with Generalized Cable Selection Rule . . . . .	57
3.1	FAS-Pivot Algorithm . . . . .	74
3.2	CC-Pivot Algorithm . . . . .	86
3.3	CC-RepeatChoice Algorithm . . . . .	91
3.4	FASLP-Pivot Algorithm . . . . .	104
3.5	Derandomization of FASLP-Pivot. . . . .	109
3.6	CCLP-Pivot Algorithm . . . . .	112
3.7	Algorithm for Hierarchical Clustering . . . . .	117
4.1	CPU Time versus Performance on Web Search Data . . . . .	152
4.2	CPU Time versus Performance on Web Communities Data . . . . .	153
4.3	CPU Time versus Performance on Web Communities Data, Zoomed in . . . . .	154
4.4	Best Result after Repeated Runs of the QuickSort Algorithms on Web Search Data . . . . .	156
4.5	Best Result after Repeated Runs of the QuickSort Algorithms on Web Communities Data . . . . .	157
4.6	95% Confidence Intervals for Best Result after Repeated Runs of the QuickSort Algorithms on Web Search Data . . . . .	158

# CHAPTER 1

## INTRODUCTION

It is an outstanding fundamental question whether everything computable in randomized polynomial time is also computable in deterministic polynomial time. In other words, does the ability to make random choices give any additional computational power with respect to polynomial time? The recent PRIMES in P result by Agrawal, Kayal, and Saxena [1] provided some additional evidence that it does not; however there is still no answer to this question. In this dissertation, we consider a number of randomized algorithms for NP-hard optimization problems that provide solutions that come with a guarantee on the expected quality of the solution. Prior to this work, it was unknown whether there exist deterministic algorithms with the same guarantees. We show for almost all of the algorithms that we consider that such deterministic algorithms do indeed exist.

We begin by defining more precisely what we mean by an “algorithm that provides solutions that come with a guarantee on the quality of the solution”. Many real-world optimization problems are NP-hard: If  $P \neq NP$ , which is widely believed to be the case, then there do not exist polynomial-time algorithms that are guaranteed to find the optimal solution. By polynomial-time we mean that the running time of the algorithm is bounded by a polynomial in the size of the input. Nonetheless, we need to find solutions for these problems, so we can either use an algorithm that is not polynomial-time, meaning that for reasonably sized instances, the running time might become unacceptably large, or give up on trying to find optimal solutions, and try to find good solutions instead. An approximation algorithm does the latter: it is a polynomial-time algorithm which finds a solution that is provably good. More precisely, an  $\alpha$ -approximation algorithm for a minimization problem is an algorithm that runs in polynomial time and produces a solution for which the objective value is at most a factor  $\alpha$  times the op-

timal value. The *performance guarantee*  $\alpha$  of an approximation algorithm is thus (an upper bound on) the worst case ratio of the objective value of the solution returned by the algorithm, and the objective value of an optimal solution. A *randomized* algorithm is an algorithm that uses random numbers to determine some choices made during its execution. A randomized  $\alpha$ -approximation algorithm is then a randomized algorithm that runs in polynomial time and produces a solution for which the *expectation* of the objective value is within a factor  $\alpha$  of the optimal value, where the expectation is taken with respect to the random numbers used by the algorithm. We will refer to an algorithm that is not a randomized algorithm as a deterministic algorithm.

In this dissertation we consider several optimization problems that are NP-hard, but for which good randomized approximation algorithms are known. We show how to give deterministic variants of these algorithms that have similar performance guarantees.

We can think of the design and analysis of an approximation algorithm for a particular optimization problem as a game with two players. Given the set of all possible inputs to the problem, the first player proposes an algorithm that finds a solution given any input. The opponent sees the algorithm, and chooses the worst possible input for that algorithm. The outcome of the game is the ratio between the objective value of the algorithm's solution, and the optimal value for that input.

If the first player is allowed to propose a randomized algorithm, then the solution of the algorithm depends not only on the input but also on the outcome of the random choices made by the algorithm. The opponent again chooses the worst possible input, but now the outcome of the game is the ratio between the expectation (over the random choices made by the algorithm) of the objective value of the algorithm's solution, and the optimal value for that input. Intuitively, it seems that it is easier for the first player to do well if he or she is allowed to propose a randomized algorithm.

This game-theoretic perspective on randomized algorithms is called *foiling the adversary* in Karp’s lectures on randomized algorithms [51]. Karp gives a number of general principles that underlie almost all randomized algorithms (see also the preface of the book on randomized algorithms by Motwani and Raghavan [58]). We mention a subset of these principles, which are particularly relevant for the randomized algorithms we consider in this dissertation, namely *random sampling*, *load balancing* and *random reordering*.

*Random sampling* is one of the key ideas underlying the randomized algorithms we consider in Chapter 2. The idea behind random sampling is that a subset of the input will give a good representation of the entire input. As an example, one of the problems considered in Chapter 2 is the single source rent-or-buy problem. Given a graph  $G = (V, E)$ , with nonnegative edge costs  $c_e$  for  $e \in E$ , a source  $s \in V$  and sinks  $t_1, \dots, t_k$  with demands  $d_1, \dots, d_k$  and a parameter  $M > 1$ , the goal is to install capacity on the edges so that we can simultaneously satisfy all demands from the source. We can either rent an edge  $e$ , in which case we pay  $c_e$  per unit of demand passing through  $e$ , or we can buy the edge so it can support any amount of demand at a fixed cost  $Mc_e$ . The Sample-Augment algorithm proposed by Gupta, Kumar, Pál and Roughgarden [43, 42] for the single source rent-or-buy problem randomly chooses a subset of the sinks of expected size  $\frac{\sum_{i=1}^k d_i}{M}$ , buys edges to connect these sinks to the source and augments this solution by renting edges to connect the remaining sinks to the source. If the random sample indeed gives a good representation of the entire input, then the set of edges we buy to connect the random sample should be useful also for the sinks that we did not sample.

The second principle that can help to understand why the Sample-Augment algorithm gives a good solution is *load balancing*. Load balancing is the idea that randomly

assigning load to resources will evenly “spread” the load on the different resources. Although the problems we consider here are not load balancing problems, we can view the design and analysis of the Sample-Augment algorithm itself as a load balancing problem: a sink incurs either a buying cost (if it is contained in the random sample) or a renting cost (if it is not contained in the random sample). Gupta et al. [43, 42] demonstrate a sampling strategy that balances the sinks’ expected renting and buying costs. It turns out not to be too difficult to bound the expected buying cost in terms of the optimal value, and hence by balancing the two costs, they also obtain a bound on the expected renting cost.

The third principle of randomized algorithms we mention here is *random reordering*, which is the main principle underlying the algorithms in Chapter 3. The idea is that the algorithm will perform well on most instances, except if it makes its decisions in an “unlucky” order. So after randomly reordering the input, and then applying the algorithm, the output will be very likely to be good. In Chapter 3 we consider the “pivoting” algorithms for ranking and clustering that were proposed by Ailon, Charikar and Newman [4]. A pivoting algorithm is a recursive algorithm, which chooses an element as pivot, greedily orders the other elements with respect to this element, and then recursively orders the other elements. In the case of ranking this means we decide whether to order each element before or after the pivot, and then recurse on the elements ranked before the pivot and the elements ranked after the pivot. In the case of clustering we decide whether or not to order the element in the same cluster as the pivot, and recurse on the elements that are not in the same cluster as the pivot. Random reordering in this context means that the pivot is chosen uniformly at random from the elements in the recursive call.

With one exception, no derandomizations were known for the algorithms considered in this dissertation. The exception is the Sample-Augment algorithm for single source rent-or-buy in Chapter 2 that we discussed above. Gupta, Srinivasan and Tardos [46] derandomize this algorithm using the following idea. Rather than sampling the sinks independently at random, the sinks are sampled with limited dependence. Gupta et al. show that under this sampling strategy, the Sample-Augment algorithm is a 4.2-approximation algorithm. Then, since this sampling strategy has a small sample space, the algorithm can be derandomized by considering all points in the sample space.

Although randomized algorithms are very powerful, and often have the advantages of being fast and simple, there are reasons for preferring deterministic algorithms. In the case of approximation algorithms, the performance guarantee of a randomized algorithm is a guarantee on the expectation of the algorithm's solution only. It provides no information about the variance of the solution's objective value. Even if the variance is small, it is often undesirable to get different solutions for the same input. And finally, as we already mentioned, it is an outstanding question of theoretical interest whether everything computable in randomized polynomial time is also computable in deterministic polynomial time.

In this dissertation we give deterministic variants for the two classes of algorithms mentioned above. The idea behind the derandomization of the pivoting algorithms in Chapter 3 is very simple: instead of randomly choosing an element as pivot (or the principle of random reordering from above), we choose a good pivot (or a good ordering of the input). To be able to do this, we present an alternative, and arguably simpler, proof of the performance guarantees of the randomized algorithms, which immediately gives us a way of choosing a good pivot, and hence deterministic algorithms.

The key tool used for the derandomization of the Sample-Augment algorithms in Chapter 2 is a linear programming relaxation of the problem. Intuitively, rather than relying on random sampling to provide a representative sample of the entire input, we can use a linear programming relaxation to give us a global view of the input and help us choose a good sample.

What will be clear from these two chapters is that to obtain a deterministic algorithm that is as powerful as the randomized algorithm, we will often require significantly more computation. In theory this is not a problem, since the algorithms we propose run in polynomial time, but practically speaking this is not desirable. In Chapter 4, we consider an important application of the ranking problem we consider in Chapter 3. The rank aggregation problem is the problem of finding a ranking a set of elements that best represents a given set of input rankings of the elements. This is a classical problem from social choice and voting theory, but there has been a lot of interest in this problem in the computer science community in recent years. One application we consider is that of building meta-search engines for Web search, where we want to combine the ranked search results obtained by different algorithms into a representative ranking of search results.

We perform an extensive evaluation of several known and new algorithms for rank aggregation on web search data, including the randomized and deterministic algorithms from Chapter 3. We argue that there are two important classes of algorithms for rank aggregation: positional methods and comparison sort methods. We propose new ways of combining these two approaches into what we call “hybrid” algorithms. We find that, although all algorithms perform significantly better than their theoretical performance guarantees, these hybrid algorithms give deterministic algorithms with a particularly good trade-off in running time and performance.



Chapters 2 and 3 are joint work with David P. Williamson. Chapter 4 is joint work with Frans Schalekamp.

## CHAPTER 2

# DETERMINISTIC SAMPLE-AUGMENT ALGORITHMS FOR NETWORK DESIGN PROBLEMS

## 2.1 Introduction

In this chapter, we consider a number of different network design problems: they feature an underlying undirected graph  $G = (V, E)$  with edge costs  $c_e \geq 0$  that satisfy the triangle inequality, and we need to make decisions such as on which edges to install how much capacity or at which vertices to open facilities. For the problems we consider, the best known approximation algorithms are remarkably simple randomized algorithms. The algorithms randomly mark a subset of the vertices, solve a certain subproblem on the random sample, and augment the solution for the subproblem to a solution for the original problem. Following [42], we will refer to this type of algorithm as a Sample-Augment algorithm. We give a general framework that allows us to derandomize most Sample-Augment algorithms, i.e. to specify a specific sample for which the cost of the solution created by the Sample-Augment algorithm is at most a constant factor away from optimal. Our approach allows us to obtain the best deterministic approximation algorithms for the single source rent-or-buy problem, the connected facility location problem, in which the open facilities need to be connected by either a tree or a tour [22], the virtual private network design problem [43, 42, 20, 21], 2-stage stochastic Steiner tree problem with independent decisions [44], the *a priori* traveling salesman problem [62], and even the single-sink buy-at-bulk problem [43, 42, 39], although for this we need to further extend our framework. We defer definitions of the problems we consider to the relevant sections. We refer the reader also to the paper by Gupta, Kumar, Pál and Roughgarden [42], which is the journal version of the papers which first introduced

Sample-Augment algorithms [43, 41].

As an example, in the single source rent-or-buy problem, we are given a source  $s \in V$ , a set of sinks  $t_1, \dots, t_k \in V$  and a parameter  $M > 1$ . An edge  $e$  can either be *rented* for sink  $t_j$  in which case we pay  $c_e$ , or it can be bought and used by any sink, in which case we pay  $Mc_e$ . The goal is to find a minimum cost set of edges to buy and rent so that for each sink  $t_j$  the bought edges plus the edges rented for  $t_j$  contain a path from  $t_j$  to  $s$ . In the Sampling Step of the Sample-Augment algorithm in Gupta et al. [43, 42] we mark each sink independently with probability  $\frac{1}{M}$ . Given the set of marked sinks  $D$ , the Subproblem Step finds a Steiner tree on  $D \cup \{s\}$  and buys the edges of this tree. In the Augmentation Step, the subproblem's solution is augmented to a feasible solution for the single source rent-or-buy problem by renting edges for each unmarked sink  $t_j$  to the closest vertex in  $D \cup \{s\}$ .

To give a deterministic version of the Sample-Augment algorithm, we want to find a set  $D$  such that for this set  $D$  the cost of the Subproblem Step plus the Augmentation Step is at most the expected cost of the Sample-Augment problem. One natural approach is to try and use the method of conditional expectations [23] to achieve this. However, in order to do this we would need to be able to compute the conditional expectation of the cost of the Sample-Augment problem, conditioned on including / not including  $t_j \in D$ . Unfortunately, we do not know how to do this for any of the problems for which good Sample-Augment algorithms exist.

We will see however that we can get around this problem by using a good upper bound to provide an estimate of the conditional expectations required. We give more details behind our approach in Subsection 2.1.2, but first discuss some related work.

### 2.1.1 Related Work

Sample-Augment algorithms were first introduced by Gupta, Kumar and Roughgarden [43]. They use the framework to give new approximation algorithms for the single source rent-or-buy, virtual private network design and single-sink buy-at-bulk problems. The main principle behind the analysis of the Sample-Augment algorithms is that under the right sampling strategy (i) it is not too difficult to bound the expected subproblem cost in terms of the optimal cost, and (ii) the expected augmentation cost is bounded by the expected subproblem cost.

Gupta, Kumar, Pál and Roughgarden [42] extend this framework, and show how to obtain an improved constant factor approximation algorithm for the multicommodity rent-or-buy problem. The key new ingredient is the notion of cost shares. If  $D$  is the set of marked vertices in the Sample-Augment algorithm, then a cost sharing method gives a way of allocating the cost of the subproblem's solution on  $D$  to the vertices in  $D$ . By imposing a so-called strictness requirement on the cost sharing method, they ensure that the expected cost incurred for vertex  $j$  in the augmentation step is approximately equal to the  $j$ 's expected cost share. It is again not difficult to bound the expected cost of the subproblem in terms of the optimal cost, and hence the strictness of the cost shares implies that we can also bound the expected augmentation cost.

The ideas of strict cost shares and sampling algorithms has since then also been successfully generalized and applied to give approximation algorithms for certain *stochastic* optimization problems. The Boosted Sampling algorithm for two-stage stochastic optimization problems was introduced by Gupta, Pál, Ravi and Sinha [44], and it was extended to multi-stage stochastic optimization problems by the same authors in [45].

As an example, consider the two-stage rooted stochastic Steiner tree problem, of

which we will consider a special case in Subsection 2.2.2. Given a graph  $G = (V, E)$  with edge costs  $c_e \geq 0$ , we are given a root  $s$  and terminals  $t_1, \dots, t_k$  and a parameter  $\sigma > 1$ . A solution can be constructed in two stages. In the first stage we do not know which terminals need to be connected to the root, and we can buy edges at cost  $c_e$ . In the second stage, we do know which terminals need to connect to the root (we will call these *active*) and we can buy edges at cost  $\sigma c_e$ . We assume the probability distribution from which the set of active terminals is known, either explicitly or as a black box from which we can sample. Examples of explicit probability distributions that have been considered in the literature are the case when there is a polynomial number of possible scenarios or the case when terminals are active independently with known probabilities. The Boosted Sampling algorithm is very similar to the Sample-Augment algorithms: we draw a random sample from the terminals, we buy a Steiner tree on these vertices in the first stage, and then we augment the solution in the second stage to connect the active terminals. However the sampling distribution according to which we sample terminals is now determined by the given probability distribution on the terminals.

In summary, the simple ideas underlying the Sample-Augment algorithms and Boosted Sampling algorithms have given rise to the best approximation algorithms for a great variety of problems. We refer the reader to the relevant sections below for references for the best known sampling algorithms for the problems we consider.

For most of the problems we consider in this chapter, no constant factor *deterministic* approximation algorithms were known. The three exceptions are the connected facility location problem in which the open facilities need to be connected by a tree, the single-sink buy-at-bulk problem and the single source rent-or-buy problem: Swamy and Kumar [64] give a primal-dual 8.55-approximation algorithm for the connected facility location problem. Their analysis was recently refined to give a slightly better

approximation guarantee of 8.29 [48]. Talwar [65] gives a constructive proof that a linear programming relaxation of the single-sink buy-at-bulk problem introduced by Garg, Khandekar, Konjevod, Ravi, Salman and Sinha [33] has an integrality gap of at most 216. Finally, the Sample-Augment algorithm for the single source rent-or-buy problem is the only sampling algorithm that had been derandomized prior to our work. Gupta, Srinivasan and Tardos [46] derandomize this algorithm using the following idea. Rather than sampling the sinks independently at random, the sinks are sampled with limited dependence. Gupta et al. show that under this sampling strategy, the Sample-Augment algorithm is a 4.2-approximation algorithm. Then, since this sampling strategy has a small sample space, the algorithm can be derandomized by considering all points in the sample space.

We note that Goyal, Gupta, Leonardi and Ravi [38] recently proposed a primal-dual 8-approximation algorithm for the rooted stochastic Steiner tree problem with a polynomial number of scenarios. However, in Section 2.2.2 we consider the version of the problem with independent decisions, for which no deterministic constant factor approximation algorithm was known.

### 2.1.2 Our Results

We give deterministic versions of the Sample-Augment algorithms: in particular, we show how to find a subset of the vertices  $D$  such that for this set  $D$  the cost of the Subproblem Step plus the Augmentation Step is at most the expected cost of the Sample-Augment problem.

Our approach is based on the method of conditional expectations [23]. We iterate through the vertices and decide whether or not to include the vertex in  $D$  depending

on which choice gives a lower expected cost. Since we do not know how to compute the conditional expectation of the cost of the Sample-Augment problem, conditioned on including / not including the vertex in  $D$ , we need to use an estimate of these conditional expectations. What we show is that we can find an upper bound on the cost of the Subproblem Step plus Augmentation Step that can be efficiently computed. In addition, we show that the expectation of the upper bound under the sampling strategy of the randomized Sample-Augment algorithm is at most  $\beta OPT$ , where  $OPT$  is the optimal value and  $\beta > 1$  is some constant. Then we can use this upper bound and the method of conditional expectation to find a set  $D$  such that the upper bound on the cost of the Subproblem Step plus the Augmentation Step is not more than the expected upper bound for the randomized Sample-Augment algorithm, and hence at most  $\beta OPT$  as well.

Our upper bound on the cost of the Subproblem Step will be obtained from a particular feasible solution to a linear programming (LP) relaxation of the subproblem. We then use well-known approximation algorithms to obtain a solution to the subproblem that comes within a constant factor of the subproblem LP. We do not need to solve the LP relaxation of the subproblem: instead we show that the optimal solution to an LP relaxation *of the original problem* defines a set of feasible solutions to the *subproblem's* LP relaxation. We note that for some of the problems we consider, for example the virtual private network design problem, this requires us to “discover” a new LP relaxation of the original problem.

Using this technique, we derive the best known deterministic approximation algorithms for the single source rent-or-buy problem, 2-stage rooted stochastic Steiner tree problem with independent decisions, the *a priori* traveling salesman problem with independent decisions, the connected facility location problem in which the open facilities need to be connected by a Steiner tree or traveling salesman tour, the virtual private net-

work design problem and the single-sink buy-at-bulk problem. We thus partially answer an open question in Gupta et al. [42] (the only problem in [42] that we do not give a deterministic algorithm for is the multicommodity rent-or-buy problem). In addition, our analysis implies that the integrality gap of an (even more) natural LP relaxation than the one considered in [33, 65] for the single-sink buy-at-bulk problem has integrality gap at most 27.72. We summarize our results in Table 2.1.2. The table uses the following abbreviations: SSRoB (single source rent-or-buy problem), 2-stage Steiner (2-stage rooted stochastic Steiner tree problem with independent decisions), apriori TSP (*a priori* traveling salesman problem with independent decisions), CFL-tree (connected facility location problem in which open facilities need to be connected by a tree), CFL-tour (connected facility location problem in which open facilities need to be connected by a tour),  $k$ -CFL-tree (connected facility location problem in which at most  $k$  facilities can be opened and the facilities need to be connected by a tree), CPND (virtual private network design problem) and SSBaB (single-sink buy-at-bulk problem). The first column contains the best known approximation guarantees for the problems, which are obtained by randomized Sample-Augment algorithms. The second column gives the previous best known approximation guarantee by a deterministic algorithm. Entries marked with \* were obtained based on the work of Williamson and Van Zuylen [68] that describes a special case of the approach in this chapter. The third column shows the approximation guarantees in this chapter.

We remark that our method is related to the method of pessimistic estimators of Raghavan [60]: Raghavan also uses an efficiently computable upper bound in combination with the method of conditional expectation to derandomize a randomized algorithm, where he first proves that the expected “cost” of the randomized algorithm is small. (We note that in the problem he considers, the cost of the algorithm is either 0 (the solution is “good”) or 1 (the solution is “bad’’)). However, in Raghavan’s work the probabilities



Table 2.1: Summary of Best Known Approximation Guarantees for the Problems Considered in Chapter 2.

Problem	randomized	prev. best deterministic	our result
SSRoB	2.92 [22]	4.2 [46], 3.28* [68, 22]	3.28
2-stage Steiner	3.55 [44]	$\log n$ [50]	8
a priori TSP	4 [62], $O(1)$ [32]	$8^*$ [62]	6.5
CFL-tree	4 [22]	8.29 [48], 4.23* [22]	4.23
$k$ -CFL-tree	6.85 [22]	6.98* [22]	6.98
CFL-tour	4.12 [22]	-	4.12
VPND	3.55 [21]	$\log n$ [28]	8.02
SSBaB	24.92 [39]	216 [65]	27.72

in the randomized algorithm depend on a solution to a linear program, but the upper bounds are obtained by a Chernoff-type bound. In our work, the probabilities in the randomized algorithm are already known from previous works, but we demonstrate *upper bounds* on the conditional expectations that depend on linear programming relaxations.

In the next section, we will give a general description of a Sample-Augment algorithm, and give a set of conditions under which we can give a deterministic variant of a Sample-Augment algorithm. In Section 2.2.1 we illustrate our method using the single source rent-or-buy problem as an example. In Sections 2.2.2, 2.2.3, 2.2.4, 2.2.5 and 2.3.1 we sketch how to obtain deterministic versions of the Sample-Augment algorithms for the 2-stage rooted stochastic Steiner tree with independent decisions, the *a priori* traveling salesman problem, connected facility location problems, the virtual private network design problem and the single-sink buy-at-bulk problem. We conclude with a brief discussion of some future directions in Section 2.4.

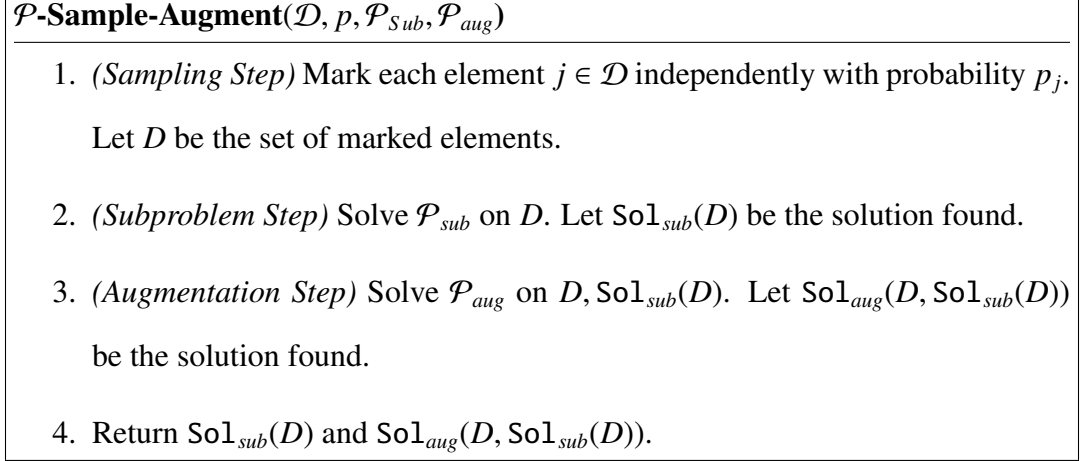


Figure 2.1: Sample-Augment Algorithm

## 2.2 Derandomization of Sample-Augment Algorithms

We give a high-level description of a class of algorithms first introduced by Gupta et al. [43], which were called Sample-Augment algorithms in [42]. Given a (minimization) problem  $\mathcal{P}$ , a Sample-Augment problem is defined by

- (i) a set of elements  $\mathcal{D} = \{1, \dots, n\}$  and sampling probabilities  $p = (p_1, \dots, p_n)$ ,
- (ii) a subproblem  $\mathcal{P}_{sub}(D)$  defined for any  $D \subset \mathcal{D}$ , and
- (ii) an augmentation problem  $\mathcal{P}_{aug}(D, \text{Sol}_{sub}(D))$  defined for any  $D \subset \mathcal{D}$  and solution  $\text{Sol}_{sub}(D)$  to  $\mathcal{P}_{sub}(D)$ .

The Sample-Augment algorithm samples from  $\mathcal{D}$  independently according to the sampling probabilities  $p$ , solves the subproblem and augmentation problem for the random subset, and returns the union of the solutions given by the subproblem and augmentation problem. We give a general statement of the Sample-Augment algorithm in Figure 2.1.

We remark that we will consider Sample-Augment algorithms in which the Augmentation Step only depends on  $D$ , and not on  $\text{Sol}_{sub}(D)$ .

In the following, we let  $OPT$  denote the optimal cost of the problem we are considering. Let  $C_{sub}(D)$  be the cost of  $\text{Sol}_{sub}(D)$ , and let  $C_{aug}(D)$  be the cost of  $\text{Sol}_{aug}(D, \text{Sol}_{sub}(D))$ . Let  $C_{SA}(D) = C_{sub}(D) + C_{aug}(D)$ . We will use blackboard bold characters to denote random sets. For a function  $C(D)$ , let  $\mathbb{E}_p[C(\mathbb{D})]$  be the expectation of  $C(\mathbb{D})$  if  $\mathbb{D}$  is obtained by including each  $j \in \mathcal{D}$  in  $\mathbb{D}$  independently with probability  $p_j$ .

Note that, since the elements are included in  $\mathbb{D}$  independently, the conditional expectation of  $\mathbb{E}_p[C_{SA}(\mathbb{D})]$  given that  $j$  is included in  $\mathbb{D}$  is  $\mathbb{E}_{p, p_j \leftarrow 1}[C_{SA}(\mathbb{D})]$ , and the conditional expectation, given that  $j$  is not included in  $\mathbb{D}$  is  $\mathbb{E}_{p, p_j \leftarrow 0}[C_{SA}(\mathbb{D})]$ . By the method of conditional expectations [23], one of these conditional expectations has value at most  $\mathbb{E}_p[C_{SA}(\mathbb{D})]$ . Hence if we could compute the expectations for different vectors of sampling probabilities, we could iterate through the elements and transform  $p$  into a binary vector (corresponding to a deterministic set  $D$ ) without increasing  $\mathbb{E}_p[C_{SA}(\mathbb{D})]$ .

Unfortunately, this is not very useful to us yet, since it is generally not the case that we can compute  $\mathbb{E}_p[C_{SA}(\mathbb{D})]$ . However, as we will show, for many problems and corresponding Sample-Augment algorithms, it is the case that  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  can be efficiently computed for any vector of probabilities  $p$ , and does not depend on the solution  $\text{Sol}_{sub}(\mathbb{D})$  for the subproblem, but only on the set  $\mathbb{D}$ . The expected cost of the subproblem's solution is more difficult to compute. What we therefore do instead is replace the cost of the subproblem by an upper bound on its cost: Suppose there exists a function  $U_{sub} : 2^{\mathcal{D}} \rightarrow R$  such that  $C_{sub}(D) \leq U_{sub}(D)$  for any  $D \subset \mathcal{D}$ , and suppose we can efficiently compute  $\mathbb{E}_p[U_{sub}(\mathbb{D})]$  and  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  for any vector  $p$ . If there exists some

vector  $\hat{p}$  such that

$$\mathbb{E}_{\hat{p}}[U_{sub}(\mathbb{D})] + \mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] \leq \beta OPT, \quad (2.1)$$

then we can use the method of conditional expectation to find a set  $D$  such that  $U_{sub}(D) + C_{aug}(D) \leq \beta OPT$ , and hence also  $C_{sub}(D) + C_{aug}(D) \leq \beta OPT$ .

In particular, the upper bounds that we will consider will all be given by solutions to an LP relaxation of the subproblem.

**Theorem 1** *Given a minimization problem  $\mathcal{P}$  and an algorithm  $\mathcal{P}$ -Sample-Augment, suppose the following four conditions hold:*

- (i)  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  depends only on  $\mathbb{D}$ , not on  $\text{Sol}_{sub}(\mathbb{D})$ , and can be efficiently computed for any  $p$ .
- (ii) There exists an LP relaxation  $\text{Sub-LP}(D)$  of  $\mathcal{P}_{sub}(D)$  and an algorithm for  $\mathcal{P}_{sub}(D)$  that is guaranteed to output a solution to  $\mathcal{P}_{sub}(D)$  that costs at most a factor  $\alpha$  times the cost of any feasible solution to  $\text{Sub-LP}(D)$ .
- (iii) There exist known vectors  $b$  and  $r(j)$  for  $j = 1, \dots, n$  such that  $y(D) = b + \sum_{j \in D} r(j)$  is a feasible solution to  $\text{Sub-LP}(D)$  for any  $D \subset \mathcal{D}$ .
- (iv) There exists a vector  $\hat{p}$  such that

$$\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq \beta OPT,$$

where  $C_{LP}(y(D))$  is the objective value of  $y(D)$  for  $\text{Sub-LP}(D)$ .

Then there exists a deterministic  $\beta$ -approximation algorithm for  $\mathcal{P}$ .

**Proof:** Let  $U_{sub}(D) = \alpha C_{LP}(y(D))$ . If we use the algorithm from (ii) in the Subproblem Step of  $\mathcal{P}$ -Sample-Augment, then by (ii),  $C_{sub}(D) \leq U_{sub}(D)$ . By (iii)

$\mathbb{E}_p[U_{sub}(\mathbb{D})]$  can be efficiently computed for any  $p$ , and by (iv) Equation (2.1) is satisfied. Hence we can use the method of conditional expectation to find a set  $D$  such that  $C_{sub}(D) + C_{aug}(D) \leq U_{sub}(D) + C_{aug}(D) \leq \beta OPT$ .  $\square$

In many cases, (i) is easily verified. In the problems we are considering here, the subproblem looks for a Steiner tree or a traveling salesman tour. It was shown by Goemans and Bertsimas [37] that the cost of the minimum cost spanning tree is at most twice the optimal value of the Steiner tree LP relaxation, and hence the minimum cost spanning tree costs at most twice the objective value of any feasible solution to this LP. For the traveling salesman problem, it was shown by Wolsey [67], and independently by Shmoys and Williamson [63], that the Christofides algorithm [15] gives a solution that comes within a factor of 1.5 of the subtour elimination LP. The solution  $y(D) = b + \sum_{j \in D} r(j)$  will be defined by using the optimal solution to an LP relaxation *of the original problem*, so that for appropriately chosen probabilities  $\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$  is bounded by a constant factor times  $OPT$ . Using the analysis for the randomized algorithm to bound  $\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})]$ , we can then show that (iv) holds.

**Remark 1** *In some cases,  $\mathcal{P}_{sub}$  and  $\mathcal{P}_{aug}$  are only defined for  $D \neq \emptyset$ . In such cases, we require that condition (i) holds for all  $p$  such that  $p_j = 1$  for some  $j$ , and that condition (ii) holds for non-empty subsets  $D$ . Condition (iv) then asks for  $\hat{p}$  such that  $\hat{p}_j = 1$  for some  $j$ . The derandomization procedure will not change this element, so that the Sample-Augment algorithm is always well defined for the vectors  $p$  that we consider.*

In the remainder of this section, we show how Theorem 1 gives the results in Table 2.1 We will use the following notation. Given an undirected graph  $G = (V, E)$  with edge costs  $c_e \geq 0$  for  $e \in E$ , we denote by  $\ell(u, v)$  the length of the shortest path from  $u \in V$  to  $v \in V$  with respect to costs  $c$ . For  $S \subseteq V$  we let  $\ell(u, S) = \min_{v \in S} \ell(u, v)$ . For  $T \subseteq E$ , we

will use the short hand notation  $c(T)$  for  $\sum_{e \in T} c_e$  for  $T \subseteq E$ . Finally, for a subset  $S \subset V$ , we let  $\delta(S) = \{\{i, j\} \in E : i \in S, j \in V \setminus S\}$ .

### 2.2.1 Single Source Rent-or-Buy

We illustrate Theorem 1 by showing how it can be used to give a deterministic algorithm for the single source rent-or-buy problem. This is arguably the simplest application of Theorem 1 and hence provides a nice illustration of the more general approach.

In the single source rent-or-buy problem, we are given an undirected graph  $G = (V, E)$ , edge costs  $c_e \geq 0$  for  $e \in E$ , a source  $s \in V$  and a set of sinks  $t_1, \dots, t_k \in V$ , and a parameter  $M > 1$ . A solution is a set of edges  $B$  to *buy*, and for each sink  $t_j$  a set of edges  $R_j$  to *rent*, so that  $B \cup R_j$  contains a path from  $s$  to  $t_j$ . The cost of renting an edge  $e$  is  $c_e$  and the cost of buying  $e$  is  $Mc_e$ . We want to find a solution  $(B, R_1, \dots, R_k)$  that minimizes  $Mc(B) + \sum_{j=1}^k c(R_j)$ .

Gupta et al. [43] propose the random sampling algorithm given in Figure 2.2, where they set  $p_j = \frac{1}{M}$  for all  $j = 1, \dots, k$ .

Note that the expected cost of the Augmentation Step of SSRoB-Sample-Augment does not depend on the tree bought in the Subproblem Step. Gupta et al. [43] show that if each sink is marked independently with probability  $\frac{1}{M}$  then the expected cost of the Augmentation Step can be bounded by  $2OPT$ .

**Lemma 2 ([43])** *If  $p_j = \frac{1}{M}$  for  $j = 1, \dots, k$ , then  $\mathbb{E}[C_{aug}(\mathbb{D})] \leq 2OPT$ .*

**Lemma 3** *There exists a deterministic 4-approximation algorithm for SSRoB.*

**SSRoB-Sample-Augment**( $G = (V, E), c, s, \{t_1, \dots, t_k\}, p$ )

1. (*Sampling Step*) Mark each sink  $t_j$  with probability  $p_j$ . Let  $D$  be the set of marked sinks.
2. (*Subproblem Step*) Construct a Steiner tree on  $D \cup \{s\}$  and *buy* the edges of the tree.
3. (*Augmentation Step*) Rent the shortest path from each unmarked sink to the closest terminal in  $D \cup \{s\}$ .

Figure 2.2: Sample-Augment Algorithm for Single Source Rent-or-Buy

**Proof:** We verify that the four conditions of Theorem 1 hold. We begin by showing that  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$ , the expected cost incurred in the Augmentation Step, can be computed for any vector of sampling probabilities  $p$ . Consider any sink  $t \in \{t_1, \dots, t_k\}$ . We label the terminals in  $\{s, t_1, \dots, t_k\}$  as  $r_0, \dots, r_k$  such that  $\ell(t, r_0) \leq \ell(t, r_1) \leq \dots \leq \ell(t, r_k)$ . If we define  $p_s = 1$ , then the expected cost incurred for  $t$  in the Augmentation Step is

$$\sum_{i=0}^k \ell(t, r_i) p_{r_i} \prod_{j < i} (1 - p_{r_j}),$$

and  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  is the sum over these values for each  $t \in \{t_1, \dots, t_k\}$ .

Now consider the subproblem on a given subset  $D$  of  $\{t_1, \dots, t_k\}$ . From Goemans and Bertsimas [37] we know that we can efficiently find a Steiner tree on  $D \cup \{s\}$  of cost at most twice the optimal value (and hence the objective value of any feasible solution) of the following Sub-LP:

$$\begin{aligned} \min \quad & \sum_{e \in E} M c_e y_e \\ \text{(Sub-LP}(D)) \quad & \text{s.t.} \quad \sum_{e \in \delta(S)} y_e \geq 1 \quad \forall S \subset V : s \notin S, D \cap S \neq \emptyset \\ & y_e \geq 0 \quad \forall e \in E. \end{aligned}$$

We now want to define a feasible solution  $y(D)$  to Sub-LP( $D$ ) for any  $D \subset \mathcal{D}$ , such that

$y(D)$  can be written as  $b + \sum_{t_j \in D} r(j)$ , since this form will allow us to efficiently compute  $\mathbb{E}_p[C_{LP}(y(\mathbb{D}))]$ . To do this, we use an LP relaxation of the single source rent-or-buy problem. Let  $b_e$  be a variable that indicates whether we buy edge  $e$ , and let  $r_e^j$  indicate whether we rent edge  $e$  for sink  $t_j$ .

$$\begin{aligned}
& \min \quad \sum_{e \in E} M c_e b_e + \sum_{e \in E} \sum_{j=1}^k c_e r_e^j \\
(\text{SSRoB-LP}) \quad & \text{s.t.} \quad \sum_{e \in \delta(S)} (b_e + r_e^j) \geq 1 \quad \forall S \subset V : t_j \in S, s \notin S \\
& b_e, r_e^j \geq 0 \quad \forall e \in E, j = 1, \dots, k.
\end{aligned}$$

SSRoB-LP is a relaxation of the single source rent-or-buy problem, since the optimal solution to the single source rent-or-buy problem is feasible for SSRoB-LP and has objective value  $OPT$ . Let  $\hat{b}, \hat{r}$  be an optimal solution to SSRoB-LP. For a given set  $D \subset \mathcal{D}$  and edge  $e \in E$  we let

$$y_e(D) = \hat{b}_e + \sum_{t_j \in D} \hat{r}_e^j.$$

Clearly,  $y(D)$  is a feasible solution to Sub-LP( $D$ ) for any  $D$ .

Finally, we show the existence of a vector  $\hat{p}$  such that  $2\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] + \mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] \leq 4OPT$ . Let  $\hat{p}_j = \frac{1}{M}$  for every  $t_j \in \mathcal{D}$ . Then by Lemma 2, the expected cost of the Augmentation Step is at most  $2OPT$ , and  $2\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$  is

$$2 \sum_{e \in E} M c_e (\hat{b}_e + \sum_{j=1}^k \frac{1}{M} \hat{r}_e^j) \leq 2OPT.$$

Hence, applying Theorem 1, we get that there exists a 4-approximation algorithm for the single-sink rent-or-buy problem.  $\square$

We can show that a better deterministic approximation algorithm exists, by using the improved analysis of the randomized algorithm given by Eisenbrand, Grandoni, Rothvoß and Schäfer [22], which allows us to more carefully balance the charge against



the optimal renting and the optimal buying costs. For a given optimal solution, let  $B^*$  be the buying cost and  $R^*$  the renting cost. We need the following lemma from Eisenbrand et al. [22].

**Lemma 4 ([22])** *If  $p_j = \frac{q}{M}$  for  $j = 1, \dots, k$  then  $\mathbb{E}_p[C_{aug}(\mathbb{D})] \leq \frac{0.807}{q}B^* + 2R^*$ .*

Note that if we mark each  $t_j$  with probability  $\frac{q}{M}$ , then  $\mathbb{E}_p[C_{LP}(\mathbb{D})] = \sum_{e \in E} M c_e \hat{b}_e + q \sum_{e \in E} \sum_{j=1}^k c_e \hat{r}_e^j$ . We would like to claim that this is at most  $B^* + qR^*$ , but this is not necessarily the case. However, it is true if we replace the objective of SSROB-LP by

$$\min \sum_{e \in E} M c_e b_e + q \sum_{e \in E} \sum_{j=1}^k c_e r_e^j.$$

Hence if we use the optimal solution to SSROB-LP with the modified objective to define  $y(D)$ , then for  $\hat{p} = \frac{q}{M}$ , we get that

$$\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + 2\mathbb{E}_{\hat{p}}[C_{LP}(\mathbb{D})] \leq \frac{0.807}{q}B^* + 2R^* + 2B^* + 2qR^* = \left(\frac{0.807}{q} + 2\right)B^* + (2 + 2q)R^*.$$

Choosing  $q = 0.636$ , we get the following result.

**Lemma 5** *There exists a deterministic 3.28-approximation algorithm for Single Source Rent-or-Buy.*

### 2.2.2 2-Stage Stochastic Steiner Tree with Independent Decisions

The input of the 2-stage rooted stochastic Steiner tree problem with independent decisions consists of a graph  $G = (V, E)$  with edge costs  $c_e \geq 0$ , a root  $s$  and terminals  $t_1, \dots, t_k$  with activation probabilities  $q_1, \dots, q_k$  and a parameter  $\sigma > 1$ . A solution can be constructed in two stages. In the first stage we do not know which terminals need to be connected to the root, and we can install edges at cost  $c_e$ . In the second stage, we do

know which terminals need to connect to the root (we will call these *active*) and we can install edges at cost  $\sigma c_e$ . Each terminal  $t_j$  is active independently with probability  $q_j$ .

The Boosted Sampling algorithm proposed in [44] is very similar to the SSRoB-Sample-Augment algorithm. We first sample from the terminals, where terminal  $t_j$  is chosen independently with probability  $\min\{1, \sigma q_j\}$ . Let  $D$  be the set of terminals selected. The first stage solution is a Steiner tree on  $D \cup \{s\}$ . In the second stage, we augment the first stage solution by adding shortest paths from each active terminal to the closest terminal in  $D \cup \{s\}$ . We are interested in the expected cost of the algorithm's solution, and hence we can replace the Augmentation Step by adding shortest path from each terminal  $t_j$  to the closest terminal in  $D \cup \{s\}$  with edge costs  $\sigma q_j c_e$  as this gives the same expected cost. Hence the Boosted Sampling algorithm for 2-stage rooted stochastic Steiner tree problem with independent decisions is the same as the SSRoB-Sample-Augment algorithm with  $M = 1$ , except that in the Augmentation Step, the renting cost for renting edge  $e$  for terminal  $j$  is  $\sigma q_j c_e$ .

It is clear that condition (i) of Theorem 1 is again met. For condition (ii) we can use the same Sub-LP as in the previous section (with  $M = 1$ ), and we again have  $\alpha = 2$ . Now, we need a good LP relaxation to define the solutions  $y(D)$  to the Sub-LP. We claim that the optimal value of the following LP is at most  $OPT$ :

$$\begin{aligned}
 \min \quad & \frac{1}{3} \sum_{e \in E} (c_e b_e + \sum_{j=1}^k \sigma q_j c_e r_e^j) \\
 \text{(2-stage-LP)} \quad \text{s.t.} \quad & \sum_{e \in \delta(S)} (b_e + r_e^j) \geq 1 \quad \forall S \subset V : s \notin S, t_j \in S \\
 & b_e, r_e^j \geq 0 \quad \forall e \in E, j = 1, \dots, k.
 \end{aligned}$$

Suppose we could find the optimal Steiner tree on  $D \cup \{s\}$  in the Subproblem Step of the Boosted Sampling algorithm. Then Gupta et al. [42] show that the expected cost of the first stage solution is at most  $OPT$ , if the sampling probabilities are  $\min\{1, q_j \sigma\}$ . In

addition, they show that the expected cost of the second stage is at most  $2OPT$ . Hence there exists some sample  $D$  such that the cost of the optimal Steiner tree on  $D \cup \{s\}$  plus the cost of the Augmentation Step is at most  $3OPT$ . Letting  $b_e = 1$  for the first stage edges in this solution, and  $r_e^j = 1$  for the second stage edges, thus gives a solution to 2-stage-LP of cost at most  $OPT$ .

Given an optimal solution  $\hat{b}, \hat{r}$  to 2-stage-LP, we define  $y_e(D) = \hat{b}_e + \sum_{t_j \in D} \hat{r}_e^j$  as before, and taking  $\hat{p}_j = q_j$ , we find that

$$2\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] = 2 \sum_{e \in E} (c_e b_e + \sum_{j=1}^k \sigma q_j c_e r_e^j) \leq 6OPT.$$

Combining this with Gupta et al. [42]'s result that under these sampling probabilities,  $\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq 2OPT$ , Theorem 1 allows us to get the following result.

**Lemma 6** *There exists a deterministic 8-approximation algorithm for the 2-stage rooted stochastic Steiner tree problem with independent decisions.*

### 2.2.3 A Priori Traveling Salesman with Independent Decisions

In the *a priori* traveling salesman problem with independent decisions, we are given a graph  $G = (V, E)$  with edge costs  $c_e \geq 0$  and a set of terminals  $t_1, \dots, t_k$ , where terminal  $t_j$  is active independently of the other terminals with probability  $q_j$ . The goal is to find a so-called *master tour* on the set of all terminals, such that the expected cost of shortcutting the master tour to the set of active terminals is minimized.

Shmoys and Talwar [62] recently showed that a Sample-Augment type algorithm for this problem is a 4-approximation algorithm. In the Sampling Step, they randomly mark the terminals, where each terminal  $t_j$  is marked independently with probability  $p_j = q_j$ .

(If there is no  $t_j$  such that  $q_j = 1$ , then they need a revised Sampling Step to ensure at least one terminal is marked. We omit the details here.) In the Subproblem Step they find a tour on the marked terminals and finally, in the Augmentation Step they add two copies of the shortest path from each unmarked terminal to the closest marked terminal.

It is not hard to see that if at least one terminal is marked, then the Sample-Augment algorithm finds an Euler tour on the terminals, and we can shortcut the Euler tour to give the traveling salesman tour that will be the master tour.

To evaluate the expected cost of the shortcut tour on a set of active terminals  $A$ , Shmoys and Talwar upper bound the cost of shortcutting the master tour on  $A$  by assuming that for any  $S$  of size at least 2 we *always* traverse the edges found in the Subproblem Step, and we traverse the edges found in the Augmentation Step only for the active terminals. If  $|A| < 2$ , then the cost of the shortcut master tour is 0.

Since we are interested in upper bounding the expected cost of the shortcut tour, we can just consider the expectation of this upper bound. Let  $Q$  be the probability that at least 2 terminals are active, and let  $\tilde{q}_j$  be the probability that  $t_j$  is active conditioned on the fact that at least 2 terminals are active, i.e.  $\tilde{q}_j = \frac{q_j(1 - \prod_{i \neq j} (1 - q_i))}{Q}$ . The expected cost for an edge  $e$  in the tour constructed by the Subproblem Step is  $Qc_e$  and the expected cost for an edge  $e$  that is added for terminal  $j$  in the Augmentation Step is  $\tilde{q}_j c_e$ .

Hence we can instead analyze the algorithm APTSP-Sample-Augment given in Figure 2.3. We note that the vector of sampling probabilities must have at least one element set to 1, otherwise the Augmentation Step may not be well defined. We will therefore make sure that the vector  $\hat{p}$  with which we start the derandomization of APTSP-Sample-Augment has at least one element equal to 1 (in fact, it will have two elements set to 1).

**APTSP-Sample-Augment**( $G = (V, E), c, Q, \tilde{q}, s, \{t_1, \dots, t_k\}, p$ )

1. (*Sampling Step*) Mark each terminal  $t_j$  with probability  $p_j$ . Let  $D$  be the set of marked terminals.
2. (*Subproblem Step*) Construct a traveling salesman tour on  $D$ , and incur cost  $Qc_e$  for each edge on the tour.
3. (*Augmentation Step*) Add two copies of the shortest path from each unmarked terminal  $t_j$  to the closest terminal in  $D$  and incur cost  $\tilde{q}_j c_e$  for each edge.

Figure 2.3: Sample-Augment Algorithm for the *A Priori* Traveling Salesman Problem

Shmoys and Talwar [62] show that if  $\tilde{p}_j = q_j$  for every terminal, and if we were able to find a *minimum cost* solution to the subproblem, then  $\mathbb{E}_{\tilde{p}}[C_{sub}(\mathbb{D}) | |\mathbb{D}| \geq 2] \leq OPT$ , and  $\mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) | |\mathbb{D}| \geq 2] \leq 2OPT$ .

This implies that there is some non-empty set  $D^*$  such that  $C_{sub}(D^*) + C_{aug}(D^*) \leq 3OPT$ . Let  $t^*$  be one of the terminals in  $D^*$ , and set  $b_e = 1$  for each of the edges in the (minimum cost) subproblem's solution on  $D^*$ , and let  $r_e^j = 1$  for the edges added for terminal  $j$  in the Augmentation Step. Then  $b, r$  defines a feasible solution to the following LP with objective value at most  $OPT$  and hence APTSP-LP is an LP relaxation of the *a priori* Traveling Salesman Problem:

$$\begin{aligned}
 \min \quad & \frac{1}{3} \sum_{e \in E} (Qc_e b_e + \sum_{j=1}^k \tilde{q}_j c_e r_e^j) \\
 \text{(APTSP-LP)} \quad \text{s.t.} \quad & \sum_{e \in \delta(S)} (b_e + r_e^j) \geq 2 \quad \forall S \subset V : t^* \notin S, t_j \in S \\
 & b_e, r_e^j \geq 0 \quad \forall e \in E, j = 1, \dots, k.
 \end{aligned}$$

Note that we do not know  $t^*$ , but we can solve APTSP-LP for any  $t^* \in \{t_1, \dots, t_k\}$  and use the LP with the smallest objective value. Let  $\hat{b}, \hat{r}$  be an optimal solution to that LP.

We let the Sub-LP on  $D$  be

$$\begin{aligned}
(\text{Sub-LP}(D)) \quad & \min \sum_{e \in E} Q_{c_e} y_e \\
& \text{s.t.} \quad \sum_{e \in \delta(S)} y_e \geq 2 \quad \forall S \subset V : D \setminus S \neq \emptyset, D \cap S \neq \emptyset \\
& y_e \geq 0 \quad \forall e \in E.
\end{aligned}$$

Note that this satisfies condition (ii) in Theorem 1 with  $\alpha = 1.5$  by [67, 63]. To define solutions  $y(D)$  to Sub-LP( $D$ ), we let  $y_e(D) = \hat{b}_e + \sum_{t_j \in D} \hat{r}_e^j$ .

We now let  $\tilde{p}_j = q_j$  and consider the expectation of  $\mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) \mid |\mathbb{D}| \geq 2]$  and  $\mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) \mid |\mathbb{D}| \geq 2]$ . From Shmoys and Talwar we know that the second term is at most  $2OPT$ . Also, since the probability that  $t_j$  is in  $\mathbb{D}$  conditioned on  $\mathbb{D}$  having at least 2 elements is  $\tilde{q}_j$ , we get

$$\begin{aligned}
1.5\mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) \mid |\mathbb{D}| \geq 2] &= 1.5 \left( \sum_{e \in E} Q_{c_e} \hat{b}_e + \sum_{j=1}^k Q_{\tilde{q}_j c_e} \hat{r}_e^j \right) \\
&= 1.5 \sum_{e \in E} \left( Q_{c_e} \hat{b}_e + \sum_{j=1}^k q_j \left( 1 - \prod_{i \neq j} (1 - q_i) \right) c_e \hat{r}_e^j \right) \\
&\leq 1.5 \sum_{e \in E} \left( Q_{c_e} \hat{b}_e + \sum_{j=1}^k q_j c_e \hat{r}_e^j \right) \leq 4.5OPT, \quad (2.2)
\end{aligned}$$

where the last inequality holds since we showed that APTSP-LP is a relaxation of the *a priori* Traveling Salesman Problem.

Finally, we want to get rid of the conditioning on  $|\mathbb{D}| \geq 2$ . By conditioning on the two smallest indices in  $\mathbb{D}$  and then using basic properties of conditional expectation, one can show that there must exist two elements, say  $j_1 < j_2$  such that if we let  $\hat{p}_{j_1} = \hat{p}_{j_2} = 1$ ,  $\hat{p}_j = 0$  for all  $j < j_2, j \neq j_1$  and  $\hat{p}_j = q_j$  for all  $j > j_2$ , then

$$1.5\mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] + \mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] \leq 1.5\mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) \mid |\mathbb{D}| \geq 2] + \mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) \mid |\mathbb{D}| \geq 2].$$

Hence we can try all possible choices of  $j_1, j_2$ , and we will find  $\hat{p}$  with at least two

elements equal to 1, so that condition (iv) of Theorem 1 holds with  $\beta = 6.5$ . Hence we get the following result.

**Lemma 7** *There exists a deterministic 6.5-approximation algorithm for a priori Traveling Salesman Problem.*

## 2.2.4 Connected Facility Location Problems

The connected facility location problems that we consider have the following form. We are given an undirected graph  $G = (V, E)$  with edge costs  $c_e \geq 0$  for  $e \in E$ , a set of clients  $\mathcal{D} \subset V$  with demands  $d_j$  for  $j \in \mathcal{D}$ , a set of potential facilities  $\mathcal{F} \subset V$ , with opening cost  $f_i \geq 0$  for  $i \in \mathcal{F}$ , a connectivity requirement  $CR \in \{\text{Tour}, \text{SteinerTree}\}$  a parameter  $M > 1$ , and a parameter  $k > 1$ . We assume that the edge costs satisfy the triangle inequality. The goal is to find a subset of facilities  $F \subseteq \mathcal{F}$  to open and a set of edges  $T$  such that  $|F| \leq k$  ( $k$  may be  $\infty$ ) and  $T$  is a  $CR$  on  $F$  that minimizes

$$\sum_{i \in F} f_i + Mc(T) + \sum_{j \in \mathcal{D}} \ell(j, F).$$

We will say that we *buy* the edges of the set  $T$  that connect the open facilities, and that we *rent* the edges connecting each client to its closest open facility.

For ease of exposition we assume that  $d_j = 1$  for all  $j \in \mathcal{D}$ . It is not hard to adapt the analysis to the general case, as was shown in [43]. We will make a remark about this at the end of this section. In the following, we denote by  $\rho_{cr} = 1$  if  $CR = \text{SteinerTree}$  and  $\rho_{cr} = 2$  if  $CR = \text{Tour}$ , which basically indicates the requirement that any two open facilities need to be connected by  $\rho_{cr}$  edge-disjoint paths.

To determine which facilities to open, the Sample-Augment algorithm of Eisenbrand et al. [22] marks each client  $j \in \mathcal{D}$  independently with probability  $p_j$  and opens the fa-

**CFL-Sample-Augment**( $G = (V, E), c, \mathcal{D}, \mathcal{F}, f, k, M, CR$ )

1. (*Sampling Step*) Mark every client  $j$  in  $\mathcal{D}$  independently at random with probability  $p_j$ . Let  $D$  be the set of marked clients.
2. (*Subproblem Step*) Construct a  $CR$  solution on the set  $D$ . Buy the edges of this solution.
3. (*Augmentation Step*)

Compute an (approximately optimal) solution to the corresponding unconnected  $k$ -facility location problem. Let  $F_U$  be the facilities opened, and for  $j \in \mathcal{D}$  let  $\sigma_U(j)$  be the facility  $j$  is assigned to. Let  $F = \bigcup_{j \in D} \sigma_U(j)$ , and open the facilities in  $F$ .

Rent the edges from each client  $j \in \mathcal{D}$  to their closest open facility, and, in addition to the edges bought in Step 2, buy  $\rho_{cr}$  copies of the edges on the shortest path from each client  $j$  in  $D$  to its closest facility in  $F$ .

Figure 2.4: Sample-Augment Algorithm for Connected Facility Location

cilities that the marked clients are assigned to in an (approximately optimal) solution to the corresponding unconnected facility location problem. Of course, any feasible solution must have at least 1 open facility, hence we need to mark at least one client. To achieve this, Eisenbrand et al. first mark one client chosen uniformly at random. In our description of the Sample-Augment algorithm we omit this, but in our derandomization, we will make sure that the vector  $\hat{p}$  in condition (iv) of Theorem 1 has at least one element equal to 1. We give our variant of the Sample-Augment algorithm from Eisenbrand et al. [22] in Figure 2.4.

It can be verified that condition (i) of Theorem 1 is satisfied for any sampling prob-



abilities  $p$  such that  $p_j = 1$  for some  $j$ . We define Sub-LP( $D$ ) as

$$\begin{aligned}
 (\text{Sub-LP}(D)) \quad & \min \sum_{e \in E} M c_e y_e \\
 \text{s.t.} \quad & \sum_{e \in \delta(S)} y_e \geq \rho_{cr} \quad \forall S \subset V : D \setminus S \neq \emptyset, D \cap S \neq \emptyset \\
 & y_e \geq 0 \quad \forall e \in E.
 \end{aligned}$$

Condition (ii) of Theorem 1 is satisfied with  $\alpha = 2$  if  $CR = \text{SteinerTree}$  [37], or 1.5 if  $CR = \text{Tour}$  [67, 63].

Let  $\gamma = \frac{M}{|\mathcal{D}|}$ , and let  $a$  be a parameter to be determined later. We assume we know that facility  $i^*$  is open in the optimal solution. (We can drop this assumption by taking  $i^*$  to be the facility for which the following LP gives the lowest optimal value). We use the following LP to define the Sub-LP solutions. We note that this is almost an LP relaxation of the connected facility location problem, except for the weighing of the renting cost by  $(a + \gamma)\rho_{cr}$ .

$$\begin{aligned}
 (\text{CFL-LP}) \quad & \min \sum_{e \in E} M c_e b_e + (a + \gamma)\rho_{cr} \sum_{j \in \mathcal{D}} \sum_{e \in E} c_e r_e^j \\
 \text{s.t.} \quad & \sum_{e \in \delta(S)} (b_e + \rho_{cr} r_e^j) \geq \rho_{cr} \quad \forall S \subset V, i^* \notin S, j \in \mathcal{D} \cap S \\
 & r_e^j, b_e \geq 0 \quad \forall e \in E, j \in \mathcal{D}.
 \end{aligned}$$

Let  $\hat{b}, \hat{r}$  be an optimal solution to CFL-LP. Given an optimal solution to the original problem, let  $B^*, R^*$  be the total buying and renting cost. It is easily verified that the optimal value of CFL-LP is at most  $B^* + (a + \gamma)\rho_{cr}R^*$ . We define  $y_e(D) = \hat{b}_e + \rho_{cr} \sum_{j \in \mathcal{D}} \hat{r}_e^j$ .

Let  $\tilde{\mathbb{E}}_p[C(\mathbb{D})]$  denote the expectation of  $C(\mathbb{D})$  if  $\mathbb{D}$  is obtained by first choosing one client uniformly at random, and then adding each other client  $j \in \mathcal{D}$  with probability  $p_j$ . Suppose  $\tilde{p}_j = \frac{a}{M}$  for every  $j \in \mathcal{D}$ , and we mark one client chosen uniformly at random, and then mark each client with probability  $\tilde{p}_j$  as in Eisenbrand et al. [22]. Then the probability that  $j$  is marked is at most  $\frac{a}{M} + \frac{1}{|\mathcal{D}|}$ . Hence  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(\mathbb{D})] \leq B^* + (a + \gamma)\rho_{cr}R^*$ .

Depending on whether the connectivity requirement is a tour or a tree, and whether  $k$  is finite or infinite, Eisenbrand et al. [22] give different lemmas bounding the cost of the Augmentation Step.

**Lemma 8 ([22])** *Let  $k = \infty$  and  $CR = \text{SteinerTree}$ . In the Augmentation Step of CFL-Sample-Augment, use a bifactor approximation algorithm [55] that returns a solution such that  $\sum_{i \in F_U} f_i + \sum_{j \in D} \ell(j, \sigma_U(j)) \leq (1.11 + \ln \delta)O^* + (1 + \frac{0.78}{\delta})R^*$ . Then*

$$\tilde{\mathbb{E}}_{\tilde{p}}[C_{aug}(\mathbb{D})] \leq 2R^* + \frac{0.807}{a}B^* + (1 + a + \gamma) \left( (1.11 + \ln \delta)O^* + \left(1 + \frac{0.78}{\delta}\right)R^* \right).$$

**Lemma 9 ([22])** *Let  $p_j = \frac{q}{M}$  for all  $j \in \mathcal{D}$ . Let  $k < \infty$  and  $CR = \text{SteinerTree}$ , and suppose we use a  $\rho_{kfl}$ -approximation algorithm to find a solution to the unconnected  $k$ -facility location problem in the Augmentation Step of CFL-Sample-Augment, then*

$$\tilde{\mathbb{E}}_{\tilde{p}}[C_{aug}(\mathbb{D})] \leq 2R^* + \frac{0.807}{a}B^* + (1 + a + \gamma)\rho_{kfl}(R^* + O^*).$$

**Lemma 10 ([22])** *Let  $p_j = \frac{q}{M}$  for all  $j \in \mathcal{D}$ . Let  $k = \infty$  and  $CR = \text{Tour}$ . In the Augmentation Step of CFL-Sample-Augment, use a bifactor approximation algorithm [55] that returns a solution such that  $\sum_{i \in F_U} f_i + \sum_{j \in D} \ell(j, \sigma_U(j)) \leq (1.11 + \ln \delta)O^* + (1 + \frac{0.78}{\delta})R^*$ . Then*

$$\tilde{\mathbb{E}}_{\tilde{p}}[C_{aug}(\mathbb{D})] \leq 2R^* + \frac{1}{2a}B^* + (1 + 2(a + \gamma)) \left( (1.11 + \ln \delta)O^* + \left(1 + \frac{0.78}{\delta}\right)R^* \right).$$

Combining the bounds from Lemma 8, 9 and 10 with  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(\mathbb{Y}(\mathbb{D}))] \leq B^* + (a + \gamma)\rho_{cr}R^*$ , we can obtain bounds on  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{aug}(\mathbb{D})] + \alpha\tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(\mathbb{Y}(\mathbb{D}))]$ .

Eisenbrand et al. [22] show that if  $\frac{1}{\gamma} = \frac{|\mathcal{D}|}{M} < C$  for some constant  $C$ , then there exists a polynomial-time approximation scheme (PTAS) for the connected facility location problem. Hence we can assume that  $\gamma$  is very small, since we can use the PTAS for

larger values of  $\frac{M}{|\mathcal{D}|}$ . We will show below that by choosing  $a$  and  $\delta$  appropriately, we can show that  $\tilde{\mathbb{E}}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$  is at most  $4.23OPT$ ,  $6.98OPT$  and  $4.12OPT$  respectively, for the problems in Lemmas 8, 9 and 10.

Finally, we only need to remark that by the definition of expectation, there must exist some index  $j_1$  and a vector  $\hat{p}$ , such that  $\hat{p}_{j_1} = 1$  and  $\hat{p}_j = 0$  for  $j < j_1$  and  $\hat{p}_{j'} = \frac{a}{M}$  for all  $j > j_1$  such that  $\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq \tilde{\mathbb{E}}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$ . We can thus try all possible choices for  $j_1$  and choose the vector  $\hat{p}$  that gives the smallest expectation. Then by Theorem 1 we can iterate through the elements  $j > j_1$  to get a binary vector  $p$  (or, equivalently, a deterministic sample) such that  $\mathbb{E}_p[C_{aug}(\mathbb{D})] + \alpha \mathbb{E}_p[C_{LP}(y(\mathbb{D}))] \leq \tilde{\mathbb{E}}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$ .

**Lemma 11** *There exists a deterministic 4.23-approximation algorithm for  $k$ -connected facility location with  $k = \infty$  and  $CR = \text{SteinerTree}$ .*

**Proof:** By Lemma 8, and because  $\alpha = 2$ ,  $\rho_{\text{SteinerTree}} = 1$  in this case, we get that

$$\begin{aligned} & \tilde{\mathbb{E}}_{\hat{p}}[C_{aug}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq \\ & (1 + a + \gamma) \left( 3 + \frac{0.78}{\delta} \right) R^* + \left( 2 + \frac{0.807}{a} \right) B^* + (1 + a + \gamma)(1.11 + \ln \delta) O^*. \end{aligned}$$

By taking  $a = 0.362$ ,  $\delta = 7.33$  and  $\gamma$  sufficiently small, we find that this is at most  $4.23OPT$  and by the discussion above, this means that there exists a deterministic 4.23-approximation algorithm.  $\square$

**Lemma 12** *There exists a deterministic 6.98-approximation algorithm for  $k$ -connected facility location with  $k < \infty$  and  $CR = \text{SteinerTree}$ .*

**Proof:** By Lemma 9, and because  $\alpha = 2, \rho_{\text{SteinerTree}} = 1$ , we get that

$$\tilde{\mathbb{E}}_{\tilde{p}}[C_{\text{aug}}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(y(\mathbb{D}))] \leq (1+a+\gamma)(2+\rho_{kfl})R^* + \left(2 + \frac{0.807}{a}\right)B^* + (1+a+\gamma)\rho_{kfl}O^*.$$

Using a 4-approximation algorithm for the (unconnected)  $k$ -facility location problem [8] in the Augmentation Step, we have  $\rho_{kfl} = 4$ . Taking  $a = 0.1623$  and  $\gamma$  sufficiently small, we find that  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{\text{aug}}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(y(\mathbb{D}))] \leq 6.98OPT$ .  $\square$

**Lemma 13** *There exists a deterministic 4.12-approximation algorithm for  $k$ -connected facility location with  $k = \infty$  and  $CR = \text{Tour}$ .*

**Proof:** By Lemma 10, and the fact that  $\alpha = 1.5$  and  $\rho_{\text{Tour}} = 2$ , we get that

$$\begin{aligned} \tilde{\mathbb{E}}_{\tilde{p}}[C_{\text{aug}}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(y(\mathbb{D}))] &\leq \left(0.5 + (1 + 2(a + \gamma))(2.5 + \frac{0.78}{\delta})\right)R^* \\ &\quad + \left(1.5 + \frac{1}{2a}\right)B^* + (1 + 2(a + \gamma))(1.11 + \ln \delta)O^*. \end{aligned}$$

Taking  $a = 0.19084, \delta = 6.5004$  and  $\gamma$  sufficiently small, we find that  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{\text{aug}}(\mathbb{D})] + \alpha \tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(y(\mathbb{D}))] \leq 4.12OPT$ .  $\square$

Finally, we mention that the results given above can easily be extended to the case when the demands are not necessarily all equal to 1. We now let  $\gamma = \frac{M}{\sum_{j' \in \mathcal{D}} d_{j'}}$ . It is again the case that there exists a PTAS for the connected facility location problem if  $\gamma > \frac{1}{C}$  for some constant  $C$  and hence we can assume  $\gamma$  is very small [22]. Now, the first client that is chosen is client  $j$  with probability  $\frac{d_j}{\sum_{j' \in \mathcal{D}} d_{j'}}$ , and we initialize the vector of sampling probabilities by  $\tilde{p}_j = \frac{a}{M}d_j$  for all  $j \in \mathcal{D}$ . Then Lemmas 8, 9 and 10 again hold. The objective of (CFL-LP) can be changed to  $\sum_{e \in E} M c_e b_e + (a + \gamma)\rho_{cr} \sum_{j \in \mathcal{D}} d_j \sum_{e \in E} c_e r_e^j$ , and we will again have that for an optimal solution  $\hat{b}, \hat{r}$  to (CFL-LP) we have  $\sum_{e \in E} M c_e \hat{b}_e + (a + \gamma)\rho_{cr} \sum_{j \in \mathcal{D}} d_j \sum_{e \in E} c_e \hat{r}_e^j \leq B^* + (a + \gamma)\rho_{cr}R^*$ . Therefore the same definition of  $y_e(D)$  as above will ensure that  $\tilde{\mathbb{E}}_{\tilde{p}}[C_{LP}(y(\mathbb{D}))] \leq B^* + (a + \gamma)\rho_{cr}R^*$ , and hence we have the exact same inequalities that we needed to prove Lemmas 11, 12, and 13.

### 2.2.5 Virtual Private Network Design

In the virtual private network design problem, we are given a graph  $G = (V, E)$  with edge costs  $c_e \geq 0$ , and a set of demands  $\mathcal{D} \subseteq V$ . Each demand  $j \in \mathcal{D}$  has thresholds  $b_{in}(j), b_{out}(j)$  on the amount of traffic that can enter and leave  $j$ .

A feasible solution is a set of paths  $P_{ij}$  for every ordered pair  $i, j \in \mathcal{D}$  and capacity  $u_e$  on the edges so that there is sufficient capacity for any *traffic pattern*  $\{f_{ij}\}_{i,j \in \mathcal{D}}$ : For any  $\{f_{ij}\}_{i,j \in \mathcal{D}}$  such that  $\sum_i f_{ij} \leq b_{in}(j)$  and  $\sum_i f_{ji} \leq b_{out}(j)$  for every  $j \in \mathcal{D}$  we need to have sufficient capacity on the paths, i.e.  $\sum_{i:j \in P_{ij}} f_{ij} \leq u_e$  for every  $e \in E$ . The objective is to find a solution that minimizes the cost  $\sum_{e \in E} c_e u_e$  of installing capacity.

Gupta et al. [43] proposed a random sampling algorithm for the virtual private network design problem that is very similar to the algorithm for single source rent-or-buy. The algorithm and analysis were improved by Eisenbrand and Grandoni [20] and Eisenbrand, Grandoni, Oriolo and Skutella [21]. We will show how Theorem 1 can be used to derandomize the improved algorithm in [21].

As was shown by Gupta et al. [43], we assume without loss of generality that each  $j \in \mathcal{D}$  is either a *sender* ( $b_{in}(j) = 0, b_{out}(j) = 1$ ) or a *receiver* ( $b_{in}(j) = 1, b_{out}(j) = 0$ ). Let  $\mathcal{J}$  be the set of receivers, and  $\mathcal{I}$  be the set of senders. By symmetry, we assume without loss of generality that  $|\mathcal{I}| \leq |\mathcal{J}|$ .

The algorithm as described by Eisenbrand et al. [21] partitions  $\mathcal{J}$  into  $\mathcal{I}$  groups, and chooses one non-empty group, say  $D$ , at random. In the Subproblem Step, we add one unit of capacity on a Steiner tree spanning  $\{i\} \cup D$  for each sender  $i$ , and finally, in the Augmentation Step we install one unit of capacity on the shortest path from each receiver  $j$  to the closest receiver in  $D$ . It follows from the analysis in Eisenbrand et al. that if we had an algorithm for finding the *minimum cost* Steiner trees in the Subproblem

**VPN-Sample-Augment**( $G = (V, E), c, \mathcal{J}, \mathcal{I}, p$ )

1. (*Sampling Step*) Mark each receiver  $j$  independently with probability  $p_j$ . Let  $D$  be the set of marked receivers.
2. (*Subproblem Step*) For each sender  $i \in \mathcal{I}$ , construct a Steiner tree  $T(i)$  on  $D \cup \{i\}$  and add one unit of capacity to each edge of  $T(i)$ .
3. (*Augmentation Step*) Install one unit of capacity on the shortest path from each receiver  $j \in \mathcal{J}$  to the closest receiver in  $D$ .

Figure 2.5: Sample-Augment Algorithm for Virtual Private Network Design

Step, then the expected cost of the solution constructed by their algorithm is at most  $\frac{3}{1-e^{-|\mathcal{J}|/|\mathcal{I}|}} OPT$ . We will use this fact later to give an LP relaxation of the virtual private network design problem.

For our derandomization, we assume we mark each receiver with some probability  $p_j$ . We will ensure that we only consider sampling probabilities so that at least one  $p_j$  will be 1, since otherwise the Augmentation Step is not well-defined.

The VPN-Sample-Augment algorithm is described in Figure 2.5. The algorithm installs the capacities and outputs the Steiner trees found in the Subproblem Step. If  $j'$  is the receiver in  $D$  that is closest to  $j$ , then  $P_{ij}$  is obtained by concatenating the unique path from  $j'$  to  $i$  in  $T(i)$  and the shortest path from  $j$  to  $j'$ .

Eisenbrand et al. [21] also show that there exists a (deterministic)  $(1 + \frac{|\mathcal{J}|}{|\mathcal{I}|})$ -approximation algorithm. Using Theorem 1 and the ideas from Section 2.2.3 we can show that if  $|\mathcal{J}| \geq 7|\mathcal{I}|$ , then there exists a deterministic 8.02-approximation algorithm.

**Lemma 14** *There exists a deterministic 8.02-approximation algorithm for virtual private network design.*

**Proof:** It is easily verified that condition (i) of Theorem 1 holds for all  $p$  with  $p_j = 1$  for some  $j$ . The Sub-LP for condition (ii) is made up of  $|I|$  different Steiner tree LPs, and has  $\alpha = 2$  [37]:

$$\begin{aligned} \text{Sub-LP}(D) \quad & \min \sum_{e \in E} \sum_{i \in I} c_e y_e^i \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} y_e^i \geq 1 \quad \forall i \in I, \forall S \subset V : i \in S, D \cap S \neq \emptyset \\ & y_e^i \geq 0 \quad \forall i \in I, e \in E. \end{aligned}$$

Let  $\kappa = 1 - e^{-\frac{|J|}{|I|}}$ . It follows from the analysis of Eisenbrand et al. [21] that the following LP is a relaxation of the virtual private network design problem.

$$\begin{aligned} \text{(VPN-LP)} \quad & \min \frac{\kappa}{3} \sum_{e \in E} c_e \left( \sum_{i \in I} b_e^i + \sum_{j \in J} r_e^j \right) \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} (b_e^i + r_e^j) \geq 1 \quad \forall S \subset V : i \in S \cap I, j \in J \setminus S \\ & r_e^j, b_e^i \geq 0 \quad \forall e \in E, j \in J, i \in I. \end{aligned}$$

Let  $\hat{r}, \hat{b}$  be an optimal solution to VPN-LP. We let  $y_e^i(D) = \hat{b}_e^i + \sum_{j \in D} \hat{r}_e^j$ . If we include each  $j \in J$  in  $\mathbb{D}$  independently with probability  $\tilde{p}_j = \frac{1}{|I|}$ , then  $\mathbb{P}[j \in \mathbb{D} | \mathbb{D} \neq \emptyset] = \tilde{p}_j / (1 - \prod_k (1 - \tilde{p}_k)) = \tilde{p}_j / \left(1 - \left(1 - \frac{1}{|I|}\right)^{|J|}\right) \leq \tilde{p}_j / (1 - e^{-|J|/|I|}) = \frac{1}{|I|} \frac{1}{\kappa}$ . Hence

$$\begin{aligned} \mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) | \mathbb{D} \neq \emptyset] &= \sum_{e \in E} \sum_{i \in I} c_e \left( \hat{b}_e^i + \sum_{j \in J} \mathbb{P}[j \in \mathbb{D} | \mathbb{D} \neq \emptyset] \hat{r}_e^j \right) \\ &\leq \sum_{e \in E} \sum_{i \in I} c_e \hat{b}_e^i + \frac{1}{\kappa} \sum_{e \in E} \sum_{i \in I} \sum_{j \in J} \frac{1}{|I|} \hat{r}_e^j \\ &= \sum_{e \in E} \sum_{i \in I} c_e \hat{b}_e^i + \frac{1}{\kappa} \sum_{e \in E} \sum_{j \in J} \hat{r}_e^j \\ &\leq \frac{3}{\kappa^2} OPT. \end{aligned}$$

Also, it follows from the analysis in [21] that

$$\mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) | \mathbb{D} \neq \emptyset] \leq \frac{2}{\kappa} OPT.$$

Therefore

$$\mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) | \mathbb{D} \neq \emptyset] + 2\mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) | \mathbb{D} \neq \emptyset] \leq \frac{2 + 2 \times 3/\kappa}{\kappa} OPT.$$

Hence we have shown that if  $\tilde{p}_j = \frac{1}{|\mathcal{I}|}$  for all  $j \in \mathcal{J}$ , then

$$\mathbb{E}_{\tilde{p}}[C_{aug}(\mathbb{D}) | \mathbb{D} \neq \emptyset] + \mathbb{E}_{\tilde{p}}[C_{LP}(y(\mathbb{D})) | \mathbb{D} \neq \emptyset] \leq \frac{2 + 6/\kappa}{\kappa} OPT.$$

By conditioning on the smallest index in  $\mathbb{D}$ , it follows from basic properties of conditional expectation that there must exist some  $j_1$  such that for the vector  $\hat{p}$  with  $\hat{p}_{j_1} = 1$ ,  $\hat{p}_j = 0$  for  $j < j_1$  and  $\hat{p}_j = \tilde{p}_j = \frac{1}{|\mathcal{I}|}$  for  $j > j_1$ , we have that  $\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + \mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq \frac{2+6/\kappa}{\kappa} OPT$ , and since  $|\mathcal{J}| \geq 7|\mathcal{I}|$ , the right hand side is at most  $8.02OPT$ .

Hence we evaluate  $\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + \mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))]$  for all choices of  $j_1$ , and choose the index  $j_1$  that gives the smallest expectation. We are then guaranteed that  $\mathbb{E}_{\hat{p}}[C_{aug}(\mathbb{D})] + \mathbb{E}_{\hat{p}}[C_{LP}(y(\mathbb{D}))] \leq 8.02OPT$ . By Theorem 1 we can then get a deterministic algorithm (or a binary vector  $\hat{p}$ ) with cost at most  $8.02OPT$  by iterating through the indices  $j > j_1$  and updating  $\hat{p}_j$  to be either 0 or 1, depending on which choice gives the smaller expectation.  $\square$

## 2.3 Multi-Stage Sample-Augment Algorithms

Sampling algorithms have been successfully used for various multi-stage stochastic optimization problems. In a multi-stage sampling algorithm, we mark a subset of the vertices and solve a subproblem on the marked vertices *in each stage of the algorithm*. Clearly, a difficulty in derandomizing such an algorithm using our approach is that the cost incurred by the algorithm in future stages often depends on the decisions made in the current stage, and are hence difficult to get a handle on.



One multi-stage sampling algorithm where it is possible to use the techniques from Section 2.2 is the Sample-Augment algorithm for the single-sink buy-at-bulk problem [42]. The single-sink buy-at-bulk problem is similar to the single source rent-or-buy problem, but instead of just having the option of either renting or buying an edge, we now have a choice of  $K$  different cable types, where each cable type has a certain capacity and price per unit length. This algorithm has stages corresponding to the cable types, and in stage  $k$  we install cables of type  $k$  and  $k + 1$  only. The cables installed in the current stage are then used to (randomly) redistribute the demands in the network, so that the input to the next stage is not deterministic.

There are three key properties that allow us to derandomize this algorithm. First of all, it turns out that the Sampling Step of each stage does not influence the expected cost of the future stages (although it does influence its distribution). Hence we can almost directly use the techniques from Section 2.2 to derandomize the Sampling Step. Secondly, as we will see, the random redistributing of the demands has only a small sample space, so we can enumerate all possible outcomes. Thirdly, we can give an efficiently computable upper bound for the cost of future stages, hence allowing us to choose a good outcome among these possible outcomes.

### 2.3.1 Single-Sink Buy-at-Bulk Network Design

The single-sink buy-at-bulk problem is a generalization of the single source rent-or-buy problem. We are given an undirected graph  $G = (V, E)$ , edge costs  $c_e \geq 0$  for  $e \in E$ , a sink  $t \in V$  and a set of sources  $s_1, \dots, s_n \in V$  with weight  $w_j > 0$  for source  $s_j$ . We denote  $\{s_1, \dots, s_n\} = \mathcal{S}$ . In addition, there are  $K$  cable types, where the  $k$ -th cable type has capacity  $u_k$  and cost  $\sigma_k$  per unit length. The goal is to install sufficient capacity at

minimum cost so that we can send  $w_j$  units from  $s_j$  to  $t$  simultaneously. We assume without loss of generality that  $1 = u_1 < u_2 < \dots < u_K$  and that  $1 = \sigma_1 < \sigma_2 < \dots < \sigma_K$ , since if  $u_k \leq u_\ell$  and  $\sigma_k \geq \sigma_\ell$ , then we can replace each cable of type  $k$  by a cable of type  $\ell$  without increasing the cost of the solution [42].

The single source rent-or-buy problem is the special case where  $K = 2$  and  $u_1 = 1, u_2 = \infty$  and  $\sigma_1 = 1, \sigma_2 = M$ .

After a preprocessing step, the Sample-Augment algorithm proposed by Gupta et al. [42] proceeds in stages, where in the  $k$ -th stage, it will install cables of type  $k$  and  $k + 1$ . At the beginning of stage  $k$ , enough capacity has already been installed to move the weights through the cables and gather the weights into a subset of the sources, so that each source has weight either 0 or  $u_k$ . We thus think of the weights at the beginning of stage  $k$  as being concentrated in  $S_k \subset S$ , where each  $s \in S_k$  has weight  $u_k$ . The final step installs cables of type  $K$  from  $S_K$  to the sink  $t$ .

As in [42], we first scale the parameters so that each parameter  $u_k, \sigma_k$  is a factor of 2. It was shown by Guha, Meyerson and Munagala [40] that we can round each  $u_k$  down to the nearest power of 2, and each  $\sigma_k$  up to the nearest power of 2, and increase the value of the optimal solution by at most a factor 4.

Note that we may assume without loss of generality that the *rescaled parameters* satisfy that  $\frac{\sigma_{k+1}}{u_{k+1}} < \frac{\sigma_k}{u_k}$ , since we may otherwise replace each cable of type  $k + 1$  by  $\frac{u_{k+1}}{u_k}$  cables of type  $k$ . We are guaranteed that  $\frac{u_{k+1}}{u_k}$  is integer because numerator and denominator are powers of 2.

After rescaling the capacities, we are however not guaranteed that the weights  $w_j$  are integer. As in Gupta et al. [42], we use a subroutine to redistribute the weights. The subroutine takes a tree, with weights  $w_v$  on the vertices  $v \in T$ , and a parameter  $U$  and

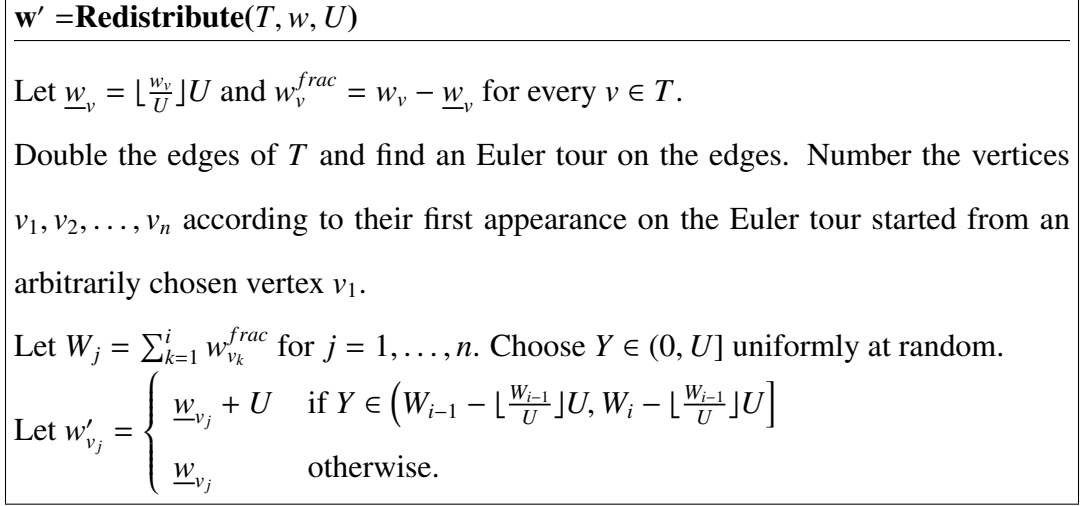


Figure 2.6: The Redistribute Subroutine

redistributes the weights along the edges of the tree, so that vertex  $v$ 's weight becomes either  $\lfloor \frac{w_v}{U} \rfloor U$  or  $\lceil \frac{w_v}{U} \rceil U$ .

In the following, we will use bold lower case letters to indicate (vectors of) random variables (and we continue to use blackboard bold capitalized letters to indicate random sets). Our description of the subroutine is given in Figure 2.6. The following lemma is a reformulation of Lemma 5.1 in [42].

**Lemma 15** [42] *Given a tree  $T$ , a parameter  $U > 0$  and weights  $w_v \geq 0$  for every  $v \in T$ , such that  $\sum_{v \in T} w_v$  is a multiple of  $U$ , let  $\underline{w}_v = \lfloor \frac{w_v}{U} \rfloor U$ . The subroutine  $\text{Redistribute}(T, w, U)$  outputs weights  $\mathbf{w}'_v$  for  $v \in T$  so that:*

- (i)  $\mathbb{P}[\mathbf{w}'_v = \underline{w}_v + U] = \frac{w_v - \underline{w}_v}{U}$ , and  $\mathbb{P}[\mathbf{w}'_v = \underline{w}_v] = 1 - \frac{w_v - \underline{w}_v}{U}$ .
- (ii) *The transportation problem instance on  $T$  where each edge in  $T$  has capacity  $U$ , and each  $v \in T$  has net demand  $\mathbf{w}'_v - w_v$  has a feasible solution with probability 1.*

In the preprocessing step, Gupta et al. [42] find a Steiner tree  $T_0$  on  $\mathcal{S} \cup \{t\}$ , install

a cable of type 1 on each edge of  $T_0$  and let  $\mathbf{w}' = \text{Redistribute}(T_0, \mathbf{w}, 1)$ . We let  $S_0 = \mathcal{S}$  and  $\mathbb{S}_1 = \{s_j \in S_0 : \mathbf{w}'_j > 0\}$ , where we note that  $\mathbb{S}_1$  is a random set, since  $\mathbf{w}'$  is a vector of random variables. For ease of exposition, we assume that  $w_j \leq 1$  for each  $s_j \in \mathcal{S}$ , so that after redistributing each source in  $S_1$  has weight 1. In Remark 2 we explain how to adapt the algorithm to the general case. At the beginning of stage  $k$ , there is sufficient capacity installed in the previous stages to move the weights from  $S_1$  to some subset  $S_k \subset S_1$  in such a way that each source in  $S_k$  gets  $u_k$  weight. We will say that “the weights are located at  $S_k$  at the start of stage  $k$ ”. We give the Sample-Augment algorithm from Gupta et al. [42] in Figures 2.7 and 2.8.

**Remark 2** *The only changes required for the general case (where we drop the assumption that  $w_j \leq 1$  for each  $s_j \in \mathcal{S}$ ) are the following. We maintain that the total weight located at source  $s_j$  at the start of stage  $k$  is an integer multiple of  $u_k$ . Let  $S_k$  again be the set of sources at which the weights are located at the start of stage  $k$ . For  $s_j \in S_k$ , let the weight located at  $s$  at the beginning of stage  $k$  be  $\omega_j^k u_k$ . We set  $\hat{p}_j^k = \omega_j^k \frac{\sigma_k}{\sigma_{k+1}}$ . In the Augmentation Step of stage  $k$ , we replace  $S_k$  by the multiset that contains  $\omega_j^k$  copies of  $s_j$ . Finally, we note that we can efficiently implement the Augmentation Step even if  $\omega_j^k$  is not polynomial in  $|V|$ .*

For ease of notation, we will refer to the preprocessing step as stage 0, the intermediate stages as stages  $1, \dots, K-1$  and the final stage as stage  $K$ .

We begin by showing that we can replace the costs incurred by the algorithm by certain upper bounds. We will see that these upper bounds allow us to bound the expected cost incurred by the algorithm, and that they will have an easy form that will help in derandomizing the algorithm. We will need several lemmas, but to keep the flow of the arguments we defer some of the proofs.

<b>SSBaB-Sample-Augment</b> ( $G = (V, E), c, t, \mathcal{S}, u_1, \dots, u_K, \sigma_1, \dots, \sigma_K$ )
<p><i>(Rounding Step)</i></p> <p>Round down all capacities <math>u_k</math> and round up all prices <math>\sigma_k</math> to the nearest power of 2.</p> <p>Remove cable type <math>k</math> if <math>\frac{\sigma_k}{u_k} \geq \frac{\sigma_{k-1}}{u_{k-1}}</math>. Let <math>K</math> be the remaining number of cable types.</p> <p><i>(Preprocessing Step)</i></p> <p>Find a Steiner tree <math>T_0</math> on <math>\mathcal{S} \cup \{t\}</math>, install a cable of type 1 on each edge of <math>T_0</math>. Let <math>w' = \text{Redistribute}(T_0, w, 1)</math>. Let <math>S_1 = \{s_j \in \mathcal{S} : w'_j &gt; 0\}</math>.</p> <p><i>(Intermediate Stages)</i></p> <p>For <math>k = 1, \dots, K - 1</math></p> $\hat{p}_j^k = \frac{\sigma_k}{\sigma_{k+1}} \text{ for } s_j \in S_k$ $S_{k+1} = \text{SSBaB-Stage-}k(G, c, t, S_k, u_k, u_{k+1}, \hat{p}^k)$ <p><i>(Final Stage)</i></p> <p>For every <math>s \in S_K</math>, install cables of type <math>K</math> on the shortest path from <math>s</math> to <math>t</math>.</p>

Figure 2.7: Sample-Augment Algorithm for Single-Sink Buy-at-Bulk

The following lemma is similar to Lemma 5.2 in [42] and will be useful throughout this section.

**Lemma 16** *For any  $k = 1, \dots, K - 1$ , let  $\mathbb{S}_k$  be the (random) set of sources at which the weights are located at the beginning of stage  $k$ . Given  $S_k \subset \mathcal{S}$  and  $k < \ell \leq K$ ,*

$$\mathbb{P}[s \in \mathbb{S}_\ell | s \in S_k] = \frac{u_k}{u_\ell}$$

*and  $\mathbb{P}[s \in \mathbb{S}_\ell | s \notin S_k] = 0$ , independent of the sampling probabilities in stage  $k$ .*

**Lemma 17** *For any  $k = 0, \dots, K$ , let  $\mathbb{S}_k$  be the random set of terminals at which the weight is located at the start of stage  $k$  (where  $\mathbb{S}_0 \equiv \mathcal{S}$ ). Let  $\mathbb{E}_{\hat{p}^k}[C_k(\mathbb{S}_k) | \mathbb{S}_k = S_k]$  be the expected cost of the cables installed in stage  $k$ , given that the weights are located in  $S_k$*

**SSBaB-stage- $k(G, c, t, S_k, u_k, u_{k+1}, p^k)$**

1. (*Sampling Step*) Mark each source  $s_j \in S_k$  independently with probability  $p_j^k$ .  
Let  $D_k$  be the set of marked sources.
2. (*Subproblem Step*) Construct a Steiner tree  $T_k$  on  $D_k \cup \{t\}$  and install cables of type  $k + 1$  on the edges of  $T_k$ .
3. (*Augmentation Step*)  
For each  $s \in S_k$ , let  $f(s) = \arg \min_{v \in D_k \cup \{t\}} \ell(s, v)$ . Install cables of type  $k$  on the shortest path from each  $s \in S_k$  to  $f(s)$ .  
For each  $v \in D_k \cup \{t\}$ , let  $S(v) = \{s \in S_k : f(s) = v\}$  and let  $w_v = \sum_{s \in S(v)} u_k$ .  
Let  $w' = \text{Redistribute}(T_k, w, u_{k+1})$ .  
For each  $v \in D_k \cup \{t\}$ , let  $S'(v)$  be a set of  $\frac{w'_v}{u_{k+1}}$  sources chosen uniformly at random from  $S(v)$ . Install a cable of type  $k + 1$  on the shortest path from  $v$  to each source in  $S'(v)$ .  
Return  $S_{k+1} = \sum_{v \in D_k \cup \{t\}} S'(v)$ .

Figure 2.8: The  $k$ -th Stage of the Sample-Augment Algorithm for Single-Sink Buy-at-Bulk

at the start of stage  $k$ , and the sampling probabilities in stage  $k$  are given by  $\hat{p}_j^k = \frac{\sigma_k}{\sigma_{k+1}}$  if  $1 \leq k \leq K - 1$ . There exist values  $B_k, R_k(j)$  for  $j = 1, \dots, n$  such that

$$\mathbb{E}_{\hat{p}^k}[C_k(\mathbb{S}_k) | \mathbb{S}_k = S_k] \leq B_k + \sum_{s_j \in S_k} R_k(j),$$

and

$$B_0 + \sum_{s_j \in S} R_0(j) + \sum_{k=1}^K \left( B_k + \sum_{s_j \in S} \frac{w_j}{u_k} R_k(j) \right) \leq 80OPT.$$

These two lemmas immediately show that the SSBaB-Sample-Augment algorithm is a *randomized* 80-approximation algorithm. This guarantee is worse than the guarantee

of 76.8 in Gupta et al. [42]. However, our analysis will be helpful in the derandomization of the algorithm.

**Corollary 18** *There exists a randomized 80-approximation algorithm for the single-sink buy-at-bulk problem.*

**Proof:** By Lemma 16 we know that  $\mathbb{P}[s_j \in \mathbb{S}_k | s_j \in S_1] = \frac{u_1}{u_k} = \frac{1}{u_k}$ , and  $\mathbb{P}[s_j \in \mathbb{S}_k | s_j \notin S_1] = 0$ , and by Lemma 15, we have that  $\mathbb{P}[s_j \in S_1] = w_j$ , hence  $\mathbb{P}[s_j \in \mathbb{S}_k] = \frac{w_j}{u_k}$ . By linearity of expectation and Lemma 17, we can thus upper bound the expected cost incurred in stage  $k$  by  $B_k + \sum_{s_j \in \mathcal{S}} \frac{w_j}{u_k} R_k(j)$ . Since  $\mathbb{P}[s_j \in S_0] = 1$  for  $s_j \in \mathcal{S}$  we can upper bound the expected cost incurred in stage 0 by  $B_0 + \sum_{s_j \in \mathcal{S}} R_0(j)$ . Hence Lemma 17 implies that the randomized algorithm in Figure 2.7 is an 80-approximation algorithm.  $\square$

Starting with our upper bound of  $B_0 + \sum_{s_j \in \mathcal{S}} R_0(j) + \sum_{k=1}^K \left( B_k + \sum_{s_j \in \mathcal{S}} \frac{w_j}{u_k} R_k(j) \right) \leq 80OPT$ , we would now would like to iterate through the random decisions made by the algorithm and turn them into deterministic decisions, without increasing the overall upper bound on the (conditional) expected cost.

The first random decisions made are those in the preprocessing step, where the Redistribute algorithm is called. This is easy to deal with because of the following lemma.

**Lemma 19** *If  $n$  is the number of vertices in  $T$ , the Redistribute subroutine on  $T$  has only  $2n + 1$  different possible outcomes.*

**Proof:** For each  $v \in T$  there is an interval  $(a_v, b_v]$  such that  $w'_v = \underline{w}_v + U$  exactly if the random variable  $Y$  is in this interval and otherwise  $w'_v = \underline{w}_v$ . If we think of the values

$a_v, b_v$  for all  $v \in T$  as points on a line  $[0, U]$  then each different outcome corresponds to a segment between two consecutive points (including the endpoints 0 and  $U$ ).  $\square$

By Lemma 19, we can consider all different outcomes of the Redistribute subroutine directly. Each outcome gives a set  $S_1$ , and by Lemma 16 we can update the upper bound on the cost as  $B_0 + \sum_{s_j \in S} R_0(j) + \sum_{k=1}^K \left( B_k + \sum_{s_j \in S_1} \frac{1}{u_k} R_k(j) \right)$ . By properties of conditional expectation, if we choose the outcome  $S_1$  for which this upper bound is smallest, we will maintain that  $B_0 + \sum_{s_j \in S} R_0(j) + \sum_{k=1}^K \left( B_k + \sum_{s_j \in S_1} \frac{1}{u_k} R_k(j) \right) \leq 80OPT$ .

The next random decisions of the SSBaB-Sample-Augment algorithm are made when marking of the sources in the Sampling Step of stage 1. Since by Lemma 16 the probability that  $s \in \mathbb{S}_k$  is  $\frac{1}{u_k}$  for  $k > 1$  and does not depend on the sampling probabilities in stage 1, we can modify the probabilities according to which we sample, and we will not change the expected upper bound on the future stages  $\sum_{k=2}^K \left( B_k + \sum_{s_j \in S_1} \frac{1}{u_k} R_k(j) \right)$ . We can thus consider only the current stage, and use a similar approach to the derandomization of single-stage Sample-Augment algorithms. We need the following lemma, which combined with Theorem 1 ensures that we can derandomize the Sampling Step of stage 1, while maintaining that the expected total cost of the cables installed in stage 1 is at most  $B_1 + \sum_{s_j \in S_1} R_1(j)$ .

**Lemma 20** *For any  $k = 1, \dots, K - 1$ , let  $B_k, R_k(j), j = 1, \dots, n$  satisfy the conditions in Lemma 17. For any  $S_k \subset S$ , the following holds for the  $k$ -stage of SSBaB-Sample-Augment:*

- (i) *The expected cost of the Augmentation Step depends only on  $\mathbb{D}_k$ , and not on the Subproblem Step, and can be efficiently computed for any  $p^k$ .*
- (ii) *There exists an LP relaxation Sub-LP $_k(D_k)$  for the minimum cost Steiner tree problem on  $D_k \cup \{t\}$  in the Subproblem Step and an algorithm for finding a Steiner tree*



on  $D_k \cup \{t\}$  that finds a solution that costs at most twice the cost of any feasible solution to  $\text{Sub-LP}_k(D_k)$ .

- (iii) There exist known vectors  $b^k$  and  $r^k(j)$  for  $j = 1, \dots, n$  such that  $y^k(D_k) = b^k + \sum_{s_j \in D_k} r^k(j)$  is a feasible solution to  $\text{Sub-LP}_k(D_k)$  for any  $D_k \subset S_k$ .
- (iv) If  $\hat{p}_j^k = \frac{\sigma_k}{\sigma_{k+1}}$  for all  $s_j \in S_k$ , then the expectation of twice the objective value of  $y^k(D_k)$  to  $\text{Sub-LP}_k(D_k)$  plus the expected cost of the cables installed in the Augmentation Step is at most  $B_k + \sum_{s_j \in S_k} R_k(j)$ .

Once we have deterministic sample  $D_1$ , the Augmentation Step of stage 1 still has two randomized processes. The first one is the Redistribute subroutine. Since  $|D_1 \cup \{t\}| \leq n + 1$ , by Lemma 19 there are at most  $2n + 3$  different outcomes of the Redistribute subroutine. Since  $D_1$  is fixed, we know  $f(s)$  for every  $s \in S_1$  and hence we also know  $w_v$  for every  $v \in D_1 \cup \{t\}$ . Each outcome of the Redistribute subroutine gives a vector  $w'$ . Since  $S'(v)$  is obtained by choosing  $\frac{w'_v}{u_2}$  sources uniformly at random from the  $\frac{w_v}{u_1}$  sources in  $S(v)$ , we know that the probability that  $s$  will be in  $\mathbb{S}_2$  is  $\frac{w'_{f(s)} u_1}{w_{f(s)} u_2}$ . Hence for each  $w'$  we can compute the conditional expectation of the cost of the type 2 cables installed in the Augmentation Step as  $\sum_{v \in D_1 \cup \{t\}} \sum_{s_j \in S(v)} \mathbb{P}[s_j \in \mathbb{S}_2 | \mathbb{D}_1 = D_1, \mathbf{w}' = w'] \sigma_2 \ell(s_j, v) = \sum_{v \in D_1 \cup \{t\}} \sum_{s_j \in S(v)} \frac{w'_{f(s_j)} u_1}{w_{f(s_j)} u_2} \sigma_2 \ell(s_j, v)$  and we can compute the expected upper bound on the cost of the remaining stages as

$$\begin{aligned} & \sum_{k=2}^K \left( B_k + \sum_{s_j \in S_1} \mathbb{P}[s_j \in \mathbb{S}_k | s_j \in \mathbb{S}_2] \mathbb{P}[s_j \in \mathbb{S}_2 | \mathbb{D}_1 = D_1, \mathbf{w}' = w'] R_k(j) \right) \\ &= \sum_{k=2}^K \left( B_k + \sum_{s_j \in S_1} \frac{u_1}{u_k} \frac{w'_{f(s_j)}}{w_{f(s_j)}} R_k(j) \right). \end{aligned} \quad (2.3)$$

We thus evaluate all  $2n + 3$  possible outcomes of  $w'$  and choose the one that gives the smallest value for the expectation of the Augmentation Step cost plus the upper bound on the cost of the remaining stages.

Finally, we can deterministically choose  $S_2$ , by iterating through the vertices  $s$  in  $S_1$  and computing the conditional expectation of the Augmentation Step plus the future stages conditioned on including / not including  $s$  in  $S_2$ . For each  $v \in D_1 \cup \{t\}$ , we let  $A_v \subset S(v)$  be the set of sources that we have already chosen to be included in  $S'(v)$  and  $B_v \subset S(v)$  the sources that we have chosen not to include in  $S'(v)$ . Initially,  $A_v = B_v = \emptyset$  for all  $v \in D_1 \cup \{t\}$ . We iterate through the sources, and compute the expected cost of the type  $k + 1$  cables in the Augmentation Step plus the expected upperbound on the cost of the remaining stages if we add  $s$  to  $A_{f(s)}$  or  $B_{f(s)}$ , and add  $s$  to the set that gives the smaller expected total cost. By the definition of conditional expectation, this does not increase the expected total cost. Since  $S_v$  contains  $\frac{w_v}{u_1}$  sources and we choose  $\frac{w'_v}{u_2}$  of these for  $S'_v$ , we can compute the conditional probability that  $s \in \mathbb{S}_2$  if  $s \notin A_{f(s)} \cup B_{f(s)}$  as

$$\mathbb{P}[s \in \mathbb{S}_2 \mid \mathbb{D}_1 = D_1, \{A_v, B_v\}_{v \in D_1 \cup \{t\}}, \mathbf{w}' = \mathbf{w}'] = \frac{w'_{f(s)}/u_2 - |A_{f(s)}|}{w_{f(s)}/u_1 - |A_{f(s)} \cup B_{f(s)}|}.$$

Hence the conditional expected cost of the future stages is

$$\sum_{k=2}^K \left( B_k + \sum_{s_j \in S_1, s_j \notin A_{f(s_j)} \cup B_{f(s_j)}} \frac{u_2}{u_k} \frac{w'_{f(s_j)}/u_2 - |A_{f(s_j)}|}{w_{f(s_j)}/u_1 - |A_{f(s_j)} \cup B_{f(s_j)}|} R_k(j) + \sum_{s_j \in S_1, s_j \in A_{f(s_j)}} \frac{u_2}{u_k} R_k(j) \right),$$

and the conditional expected cost of the type 2 cables is

$$\sum_{s_j \in S_1, s_j \notin A_{f(s_j)} \cup B_{f(s_j)}} \frac{w'_{f(s_j)}/u_2 - |A_{f(s_j)}|}{w_{f(s_j)}/u_1 - |A_{f(s_j)} \cup B_{f(s_j)}|} \sigma_2 \ell(s_j, f(s_j)) + \sum_{s_j \in S_1, s_j \in A_{f(s_j)}} \sigma_2 \ell(s_j, f(s_j)).$$

We have shown how to derandomize the preprocessing step and the first stage of SSBaB-Sample-Augment, without increasing the upper bound on the expected cost. We can use the same approach to iterate through the stages  $2, \dots, K - 1$ . We thus obtain the following result.

**Corollary 21** *There exists a deterministic 80-approximation algorithm for the single-sink buy-at-bulk problem.*

We now give the proofs of Lemma 16, 17 and 20.

**Proof of Lemma 16:** Similar to Lemma 5.2 in [42], we can prove this by induction.

If  $s \notin S_k$ , then it is clear that  $s \notin S_{k+1}$ , and hence  $s \notin S_\ell$  for any  $\ell > k$ .

Let  $s \in S_k$ , and suppose the set of marked terminals in the Sampling Step is  $D_k$ , and let  $f(s)$ ,  $w_v$ ,  $\mathbf{w}'_v$  and  $S(v)$  be defined as in the description of the  $k$ -th stage of the algorithm. Given  $\mathbf{w}'_v = w'_v$ , the probability that  $s$  is in  $\mathbb{S}_{k+1}$  is  $\frac{w'_v/u_{k+1}}{w_v/u_k} = \frac{w'_v}{w_v} \frac{u_k}{u_{k+1}}$ . By Lemma 15,  $\mathbf{w}'_v = \underline{w}_v + u_{k+1}$  with probability  $\frac{w_v - \underline{w}_v}{u_{k+1}}$  and  $\mathbf{w}'_v = \underline{w}_v$  otherwise. Hence

$$\begin{aligned} \mathbb{P}[s \in \mathbb{S}_{k+1} | s \in S_k, D_k = D_k, f(s) = v] &= \frac{\underline{w}_v}{w_v} \frac{u_k}{u_{k+1}} + \left( \frac{\underline{w}_v + u_{k+1}}{w_v} \frac{u_k}{u_{k+1}} - \frac{\underline{w}_v}{w_v} \frac{u_k}{u_{k+1}} \right) \frac{w_v - \underline{w}_v}{u_{k+1}} \\ &= \frac{u_k}{u_{k+1}}. \end{aligned} \quad (2.4)$$

Since the right hand side is the same for any  $D_k$  and  $f(s)$ , it will also hold unconditionally that  $\mathbb{P}[s \in \mathbb{S}_{k+1} | s \in S_k] = \frac{u_k}{u_{k+1}}$ .

Now, let  $s \in S_k$  and suppose the lemma holds for some  $\ell > k$ . Note that  $\mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in S_k] = \mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in \mathbb{S}_\ell] \mathbb{P}[s \in \mathbb{S}_\ell | s \in S_k] = \mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in \mathbb{S}_\ell] \frac{u_k}{u_\ell}$ , where the last equality follows from the inductive hypothesis.

Now, from (2.4) we have that for any  $S_\ell$ ,  $\mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in S_\ell] = \frac{u_\ell}{u_{\ell+1}}$ , hence for a random subset  $\mathbb{S}_\ell$  we have that  $\mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in \mathbb{S}_\ell] = \frac{u_\ell}{u_{\ell+1}}$ . Therefore  $\mathbb{P}[s \in \mathbb{S}_{\ell+1} | s \in S_k] = \frac{u_\ell}{u_{\ell+1}} \frac{u_k}{u_\ell} = \frac{u_k}{u_{\ell+1}}$ .  $\square$

We note that the (joint) distribution of  $\mathbb{S}_\ell$  *does* indeed depend on the sampling probabilities in stage  $k$ , but that by Lemma 16 the *marginal* probability  $\mathbb{P}[s \in \mathbb{S}_\ell | s \in S_k]$  does not depend on it. Because of the special form of the upper bounds and the linearity of expectation, we do not need to know the joint distribution in order to compute the expectation of the upper bound.

We will now first give the proof of Lemma 20 and then give the proof of Lemma 17.

**Proof of Lemma 20:** Given a set  $S_k$ , and sampling probabilities  $p^k$ , let  $\mathbb{E}_{p^k}[\ell(s, \mathbb{D}_k \cup \{t\})]$  be the expectation of the distance from  $s \in S_k$  to the closest terminal in  $D_k \cup \{t\}$ . By Lemma 16, the expected cost of the Augmentation Step of the  $k$ -th stage is

$$\sum_{s \in S_k} \mathbb{E}_{p^k}[\ell(s, \mathbb{D}_k \cup \{t\})](\sigma_k + \sigma_{k+1} \frac{u_k}{u_{k+1}}),$$

which does not depend on the outcome of the Subproblem Step and can be efficiently computed for any sampling probabilities  $p^k$ .

For  $k = 1, \dots, K - 1$ , the  $\text{Sub-LP}_k(D_k)$  is the linear programming relaxation of the Steiner tree problem on  $D_k \cup \{t\}$ :

$$\begin{aligned} \text{Sub-LP}_k(D_k) \quad & \min \sum_{e \in E} \sigma_{k+1} c_e y_e \\ \text{s.t.} \quad & \sum_{e \in \delta(S)} y_e \geq 1 \quad \text{for all } S \subset V, t \notin S, D_k \cap S \neq \emptyset \\ & y_e \geq 0 \quad \forall e \in E. \end{aligned}$$

By [37], we know that  $\text{Sub-LP}_k(D_k)$  satisfies the second condition of the lemma.

Similar to our previous approach, we will use a linear programming relaxation of the original problem, to show that the third condition of Lemma 20 is satisfied.

Let  $\{u_k\}_{k=1, \dots, K}, \{\sigma_k\}_{k=1, \dots, K}$  be the *rounded* cable capacities and costs that we obtain after executing the Rounding Step of SSBaB-Sample-Augment. Let  $z_e^k$  indicate whether we install a cable of type  $k$  on edge  $e$ . Let  $x_e^{j,k}$  indicate the amount of flow sent from  $s_j$

to  $t$  that passes through a cable of type  $k$  on edge  $e$ .

$$\begin{aligned}
& \min \quad \frac{1}{4} \sum_{e \in E} \sum_{k=1}^K \sigma_k c_e z_e^k \\
(\text{SSBaB-LP}) \quad & \text{s.t.} \quad \sum_{e \in \delta(S)} \sum_{k=1}^K x_e^{j,k} \geq w_j \quad \text{for all } S \subset V, j = 1, \dots, n : s_j \in S, t \notin S \\
& \sum_{j=1}^n x_e^{j,k} \leq u_k z_e^k \quad \text{for all } e \in E, k = 1, \dots, K \\
& \frac{x_e^{j,k}}{w_j} \leq z_e^k \quad \text{for all } e \in E, k = 1, \dots, K, j = 1, \dots, n \\
& x_e^{j,k} \geq 0, z_e^k \geq 0 \quad \text{for all } e \in E, k = 1, \dots, K, j = 1, \dots, n.
\end{aligned}$$

The first set of constraints enforces that  $s_j$  sends  $w_j$  units to  $t$  by enforcing that at least  $w_j$  units cross any cut that separates  $s_j$  from  $t$ . The second and third sets of constraints ensure that there is enough capacity installed to support the flow. In particular, the second set of constraints ensures that we install a sufficient number of cables of type  $k$  on edge  $e$  to support the flow sent on edge  $e$  on a type  $k$  cable jointly by all sources. The third set of constraints is implied by the second set of constraints if we constrain  $z_e^k$  to be integer, but strengthens the LP relaxation by enforcing that if source  $s_j$  sends a  $\frac{x_e^{j,k}}{w_j}$  fraction of its flow on a cable of type  $k$  on edge  $e$ , then there should be at least a  $\frac{x_e^{j,k}}{w_j}$  fraction of a type  $k$  cable installed on the edge.

To see that (SSBaB-LP) is a relaxation of the single-sink buy-at-bulk problem, consider an optimal solution to the single-sink buy-at-bulk problem. We let  $x_e^{j,k}$  be the amount of flow from  $s_j$  to  $t$  that is routed on a cable of type  $k$  on edge  $e$ . Because we rounded down the cable capacities to the nearest power of 2 to write (SSBaB-LP), we need to let  $z_e^k = 2$  for every cable of type  $k$  that is installed on edge  $e$  in this optimal solution. Since we also rounded up the cable costs to the nearest power of 2, we get that  $\sum_{e \in E} \sum_{k=1}^K \sigma_k c_e z_e^k \leq 4OPT$ .

Let  $\hat{x}, \hat{z}$  be an optimal solution to (SSBaB-LP). Now, for any stage  $k = 1, \dots, K - 1$

we define

$$b_e^k = \sum_{\ell=k+1}^K \hat{z}_e^\ell, \quad (2.5)$$

$$r_e^k(j) = \sum_{\ell=1}^k \frac{\hat{x}_e^{j,\ell}}{w_j}. \quad (2.6)$$

We let  $y_e^k(D_k) = b_e^k + \sum_{s_j \in D_k} r_e^k(j)$  for any  $D_k \subset S_k$ . We need to show that  $y_e^k(D_k)$  is feasible for  $\text{Sub-LP}_k(D_k)$  for  $k = 1, \dots, K-1$ .

Note that  $\frac{\hat{x}_e^{j,\ell}}{w_j} \leq \hat{z}_e^\ell$ , hence for any  $S \subset V$  such that  $t \in S, s_j \in D_k \setminus S$ , we have that  $\sum_{e \in \delta(S)} y_e^k(D_k) \geq \sum_{e \in \delta(S)} \left( \sum_{\ell=k+1}^K \hat{z}_e^\ell + \sum_{\ell=1}^k \frac{\hat{x}_e^{j,\ell}}{w_j} \right) \geq \sum_{e \in \delta(S)} \sum_{\ell=1}^K \frac{\hat{x}_e^{j,\ell}}{w_j} \geq 1$ . Hence  $y^k(D_k)$  is a feasible solution to the  $\text{Sub-LP}_k(D_k)$ .

Finally, we will choose  $B_k, R_k(j)$  for  $j = 1, \dots, n$  so that condition (iv) is satisfied. We will then show in the proof of Lemma 17 that this choice will also satisfy Lemma 17. Let  $C_{MST}(D_k)$  be the cost (with respect to the edge costs  $c_e$ ) of a minimum cost spanning tree on  $D_k \cup \{t\}$ . Let  $\mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k]$  be the expectation of  $C_{MST}(D_k)$  if the sampling probabilities are given by  $\hat{p}_j^k = \frac{\sigma_k}{\sigma_{k+1}}$  for  $s_j \in S_k$ , and  $\hat{p}_j^k = 0$  if  $s_j \notin S_k$ . Similarly let  $\mathbb{E}_{\hat{p}^k}[\sum_{e \in E} c_e y_e^k(\mathbb{D}_k) | \mathbb{S}_k = S_k]$  be the expectation of  $\sum_{e \in E} c_e y_e^k(\mathbb{D}_k)$ . Since  $y^k(D_k)$  is feasible for  $\text{Sub-LP}_k(D_k)$ , we know from Goemans and Bertsimas [37] that  $C_{MST}(D_k) \leq 2 \sum_{e \in E} c_e y_e^k(D_k)$  for every  $D_k \subset \mathcal{D}$ .

It follows from Lemma 5.4 in [42] that for  $k = 1, \dots, K-1$ , the expected cost of the type  $k$  cables installed in the Augmentation Step of stage  $k$ , given  $S_k$  and sampling probabilities  $\hat{p}^k$  can be bounded by

$$\left(1 - \frac{\sigma_k}{\sigma_{k+1}}\right) \sigma_{k+1} \mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k].$$

To see this, let  $X_j(D_k)$  be the cost of connecting  $s_j$  to the closest terminal in  $D_k \cup \{t\}$  using type  $k$  cables. For a given set  $D_k$ , consider the minimum cost spanning tree on  $D_k \cup \{t\}$  rooted at  $t$ , and for  $s_j \in D_k$ , let  $Y_j(D_k)$  be the length of the edge connecting  $s_j$  to

its parent in this tree, and let  $Y_j(D_k) = 0$  if  $s_j \notin D_k$ . Then  $\mathbb{E}_{\hat{p}^k}[\sum_{s_j \in S_k} Y_j(\mathbb{D}_k) | \mathbb{S}_k = S_k] = \mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k]$ . Also

$$\begin{aligned} & \mathbb{E}_{\hat{p}^k}[\sum_{s_j \in S_k} X_j(\mathbb{D}_k) | \mathbb{S}_k = S_k] \\ &= \sigma_k \sum_{s_j \in S_k} \mathbb{E}_{\hat{p}^k}[\ell(s_j, \mathbb{D}_k \cup \{t\}) | \mathbb{S}_k = S_k] \end{aligned} \quad (2.7)$$

$$= \sigma_k \sum_{s_j \in S_k} (1 - \hat{p}_j^k) \mathbb{E}_{\hat{p}^k}[\ell(s_j, \mathbb{D}_k \cup \{t\}) | \mathbb{S}_k = S_k, s_j \notin \mathbb{D}_k] \quad (2.8)$$

$$= \sigma_k \sum_{s_j \in S_k} (1 - \hat{p}_j^k) \mathbb{E}_{\hat{p}^k}[\ell(s_j, \mathbb{D}_k \setminus \{s_j\} \cup \{t\}) | \mathbb{S}_k = S_k, s_j \in \mathbb{D}_k] \quad (2.9)$$

$$\leq \sigma_k \sum_{s_j \in S_k} (1 - \hat{p}_j^k) \mathbb{E}_{\hat{p}^k}[Y_j(\mathbb{D}_k) | \mathbb{S}_k = S_k, s_j \in \mathbb{D}_k] \quad (2.10)$$

$$= \sigma_k \sum_{s_j \in S_k} (1 - \hat{p}_j^k) \frac{1}{\hat{p}_j^k} \mathbb{E}_{\hat{p}^k}[Y_j(\mathbb{D}_k) | \mathbb{S}_k = S_k] \quad (2.11)$$

$$= (1 - \frac{\sigma_k}{\sigma_{k+1}}) \sigma_{k+1} \mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k]. \quad (2.12)$$

Equation (2.7) follows from the definition of  $X_j(D_k)$ , and (2.8) follows since  $X_j(D_k) = 0$  if  $s_j \in D_k$ . Equation (2.9) follows because the sources are marked independently, so that for  $s_j \in S_k$ ,  $\mathbb{E}_{\hat{p}^k}[\ell(s_j, \mathbb{D}_k \cup \{t\}) | \mathbb{S}_k = S_k, s_j \notin \mathbb{D}_k] = \mathbb{E}_{\hat{p}^k}[\ell(s_j, \mathbb{D}_k \setminus \{s_j\} \cup \{t\}) | \mathbb{S}_k = S_k, s_j \in \mathbb{D}_k]$ . Inequality (2.10) follows since by the definition  $Y_j(D_k) \geq \ell(s_j, D_k \setminus \{s_j\} \cup \{t\})$ , and equation (2.11) follows since  $Y_j(D_k) = 0$  if  $s_j \notin D_k$ . Finally, (2.12) follows by substituting  $\hat{p}_j^k = \frac{\sigma_k}{\sigma_{k+1}}$  and  $\mathbb{E}_{\hat{p}^k}[\sum_{s_j \in S_k} Y_j(\mathbb{D}_k) | \mathbb{S}_k = S_k] = \mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k]$ .

Also, by Lemma 16, each path on which type  $k$  cables are installed in the Augmentation Step of stage  $k$  will also have type  $k+1$  cables installed in the Augmentation Step with probability  $\frac{u_k}{u_{k+1}}$ . Hence the expected cost of the type  $k+1$  cables installed in the Augmentation Step is  $\frac{\sigma_{k+1}/u_{k+1}}{\sigma_k/u_k}$  times the expected cost of the type  $k$  cables installed in the Augmentation Step. The total expected cost of the Augmentation Step of stage  $k$ , given  $S_k$  and sampling probabilities  $\hat{p}^k$  can thus be bounded by

$$\left(1 + \frac{\sigma_{k+1}/u_{k+1}}{\sigma_k/u_k}\right) \left(1 - \frac{\sigma_k}{\sigma_{k+1}}\right) \sigma_{k+1} \mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k) | \mathbb{S}_k = S_k]. \quad (2.13)$$

Now, since  $1 - \frac{\sigma_k}{\sigma_{k+1}} \leq 1$ ,  $\frac{\sigma_{k+1}/u_{k+1}}{\sigma_k/u_k} \leq \frac{1}{2}$ , and because by Goemans and Bertsimas [37]  $\mathbb{E}_{\hat{p}^k}[C_{MST}(\mathbb{D}_k)|\mathbb{S}_k = S_k] \leq 2\mathbb{E}_{\hat{p}^k}[\sum_{e \in E} c_e y_e^k(\mathbb{D}_k)|\mathbb{S}_k = S_k]$  we get that the expected cost of the Augmentation Step of stage  $k$ , given  $S_k$  and sampling probabilities  $\hat{p}^k$  is at most

$$3\sigma_{k+1}\mathbb{E}_{\hat{p}^k}[\sum_{e \in E} c_e y_e^k(\mathbb{D}_k)|\mathbb{S}_k = S_k]. \quad (2.14)$$

Because the cost of the solution created in the Subproblem Step on  $D_k$  is at most  $2\sum_{e \in E} \sigma_{k+1} c_e y_e^k(D_k)$  for any  $D_k \subset \mathcal{D}$ , we get that the total expected cost of stage  $k$  is at most  $5\sigma_{k+1}\mathbb{E}_{\hat{p}^k}[\sum_{e \in E} c_e y_e^k(\mathbb{D}_k)|\mathbb{S}_k = S_k]$ . Now, noting that  $\mathbb{E}_{\hat{p}^k}[y_e^k(\mathbb{D})|\mathbb{S}_k = S_k] = b_e^k + \sum_{s_j \in S_k} \hat{p}_j^k r_e^k(j) = b_e^k + \sum_{s_j \in S_k} \frac{\sigma_k}{\sigma_{k+1}} r_e^k(j)$ , it follows that condition (iv) of Lemma 20 is satisfied for  $k = 1, \dots, K-1$  if we let

$$B_k = 5 \sum_{e \in E} \sigma_{k+1} c_e b_e^k, \quad (2.15)$$

$$R_k(j) = 5 \sum_{e \in E} \sigma_k c_e r_e^k(j). \quad (2.16)$$

□

Finally, we show that we can define  $B_0, B_K$  and  $R_0(j), R_K(j)$  for  $j = 1, \dots, n$  so that these values combined with the values defined in (2.15, 2.16) satisfy the conditions in Lemma 17.

**Proof of Lemma 17:** We already saw that  $B_k, R_k(j)$  as defined in (2.15, 2.16) satisfy the first inequality in Lemma 17 for  $k = 1, \dots, K-1$ .

We now define  $b_e^0, r_e^K(j)$  according to the definition in (2.5), (2.6) (where we note that these definitions set  $b_e^K = 0, r_e^0(j) = 0$ ). We claim that  $b_e^0$  is a feasible solution to the Steiner LP on  $\mathcal{S} \cup \{t\}$ . Indeed consider some  $s_j \in \mathcal{S}$  and take  $S \subset V$  such that  $s_j \notin S, t \in S$ . By the third set of constraints of (SSBaB-LP),  $b_e^0 = \sum_{\ell=1}^K \hat{z}_e^\ell \geq \sum_{\ell=1}^K \frac{\hat{x}_e^\ell}{w_j}$ . Also, the first set of constraints ensures that  $\sum_{e \in \delta(S)} \sum_{\ell=1}^K \frac{\hat{x}_e^\ell}{w_j} \geq 1$ . Hence by [37], we know that we can find a tree  $T_0$  on  $\mathcal{S} \cup \{t\}$  and install cables of type 1, at cost at most  $2 \sum_{e \in E} c_e \sigma_1 b_e^0$ .



Hence we can define

$$B_0 = 2 \sum_{e \in E} \sigma_1 c_e b_e^0, \quad R_0(j) = 0, j = 1, \dots, n. \quad (2.17)$$

For the final stage, we need to give a bound on  $\sum_{s_j \in S_K} \sigma_K \ell(s_j, t)$ . From the first set of constraints of SSBaB-LP, it is clear that  $\sum_{e \in E} c_e \sum_{\ell=1}^K \frac{\hat{x}_e^{j,\ell}}{w_j}$  is an upper bound on the length of the shortest path from  $s_j$  to  $t$ , hence  $\sum_{s_j \in S_K} \sigma_K \ell(s_j, t) \leq \sum_{s_j \in S_K} \sigma_K \sum_{e \in E} c_e \sum_{\ell=1}^K \frac{\hat{x}_e^{j,\ell}}{w_j} = \sum_{e \in E} c_e \sigma_K \sum_{s_j \in S_K} r_e^K(j)$ . So we can define

$$B_K = 0, \quad R_K(j) = \sum_{e \in E} \sigma_K c_e r_e^K(j), j = 1, \dots, n. \quad (2.18)$$

Now, some algebra similar to that in Gupta et al. [42] shows that the second inequality of the lemma holds. For ease of notation, we first write  $B_K, B_0$  and  $R_K(j), R_0(j)$  in the same form as  $B_k, R_k(j)$  for  $1 \leq k \leq K-1$ . To do this, we define  $\sigma_0 = \frac{1}{2}\sigma_1, u_0 = 1, \sigma_{K+1} = 2\sigma_K, u_{K+1} = u_K$ , and increase our upper bounds  $B_0, R_0(j), B_K, R_K(j)$  so that for every  $k \in \{0, \dots, K\}$  and  $j \in \{1, \dots, n\}$ :

$$B_k = 5 \sum_{e \in E} \sigma_{k+1} c_e b_e^k, \quad R_k(j) = 5 \sum_{e \in E} \sigma_k c_e r_e^k.$$

(Note that we could have chosen any finite values for  $\sigma_0, \sigma_{K+1}$ , since  $b_e^K$  and  $r_e^0$  are 0).

Then for  $k = 1, \dots, K$ ,

$$\begin{aligned} \sum_{s_j \in S} \frac{w_j}{u_k} R_k(j) &= 5 \sum_{e \in E} \sigma_k c_e \sum_{s_j \in S} \frac{w_j}{u_k} r_e^k(j) \\ &= 5 \frac{\sigma_k}{u_k} \sum_{e \in E} c_e \sum_{s_j \in S} \sum_{\ell=1}^k \hat{x}_e^{j,\ell} \\ &\leq 5 \frac{\sigma_k}{u_k} \sum_{e \in E} c_e \sum_{\ell=1}^k u_\ell \hat{z}_e^\ell \text{ by (SSBaB-LP).} \end{aligned}$$

Also, note that  $R_0(j) = 0$ , hence  $\sum_{s_j \in S} R_0(j) = 0 \leq 5 \frac{\sigma_0}{u_0} \sum_{e \in E} c_e \sum_{\ell=1}^0 u_\ell \hat{z}_e^\ell$ .

Now, since  $B_k = 5 \sum_{e \in E} \sigma_{k+1} c_e b_e^k = 5 \sum_{e \in E} \sigma_{k+1} c_e \sum_{\ell=k+1}^K \hat{z}_e^\ell$  for  $k = 0, \dots, K$ , we get that

$$\begin{aligned}
\sum_{k=0}^K B_k + \sum_{s_j \in S} \left( R_0(j) + \sum_{k=1}^K \frac{w_j}{u_k} R_k(j) \right) &\leq 5 \sum_{k=0}^K \sum_{e \in E} c_e \left( \sum_{\ell=1}^k \sigma_k \frac{u_\ell}{u_k} \hat{z}_e^\ell + \sum_{\ell=k+1}^K \sigma_{k+1} \hat{z}_e^\ell \right) \\
&= 5 \sum_{\ell=1}^K \sum_{e \in E} c_e \hat{z}_e^\ell \left( \sum_{k=\ell}^K \sigma_k \frac{u_\ell}{u_k} + \sum_{k=0}^{\ell-1} \sigma_{k+1} \right) \\
&= 5 \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \left( \sum_{k=\ell}^K \frac{\sigma_k / u_k}{\sigma_\ell / u_\ell} + \sum_{k=1}^{\ell} \frac{\sigma_k}{\sigma_\ell} \right) \\
&= 5 \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \left( \sum_{k=0}^{K-\ell} \frac{\sigma_{\ell+k} / u_{\ell+k}}{\sigma_\ell / u_\ell} + \sum_{k=0}^{\ell-1} \frac{\sigma_{\ell-k}}{\sigma_\ell} \right) \\
&\leq 20 \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \\
&\leq 80OPT.
\end{aligned}$$

The second to last inequality follows from the fact that  $\frac{\sigma_k}{\sigma_{k+1}} \leq \frac{1}{2}$  and  $\frac{\sigma_{k+1}/u_{k+1}}{\sigma_k/u_k} \leq \frac{1}{2}$  for  $k = 0, \dots, K-1$ , since then  $\sum_{k=0}^{K-\ell} \frac{\sigma_{\ell+k}/u_{\ell+k}}{\sigma_\ell/u_\ell} \leq \sum_{k=0}^{K-\ell} \left(\frac{1}{2}\right)^k \leq 2$  and  $\sum_{k=0}^{\ell-1} \frac{\sigma_{\ell-k}}{\sigma_\ell} \leq \sum_{k=0}^{\ell-1} \left(\frac{1}{2}\right)^k \leq 2$ .

□

**Remark 3** *Our approximation guarantee is obtained with respect to the LP-relaxation SSBaB-LP. Hence we have also shown that this LP relaxation has integrality gap of at most 80. In fact, as we will show in Theorem 22, the integrality gap is at most 27.72. To the best of the author's knowledge, no previous result was known about the integrality gap of the SSBaB-LP. In [65] it was shown that a less direct LP relaxation that was proposed in [33] has an integrality gap of at most 216.*

We can improve the approximation guarantee of the algorithm using the ideas of Grandoni and Italiano [39]. Rather than rounding down the cable capacities and rounding up the prices, they carefully select a subset of the cables, which allows them to significantly improve the approximation ratio. They require the cable types to satisfy

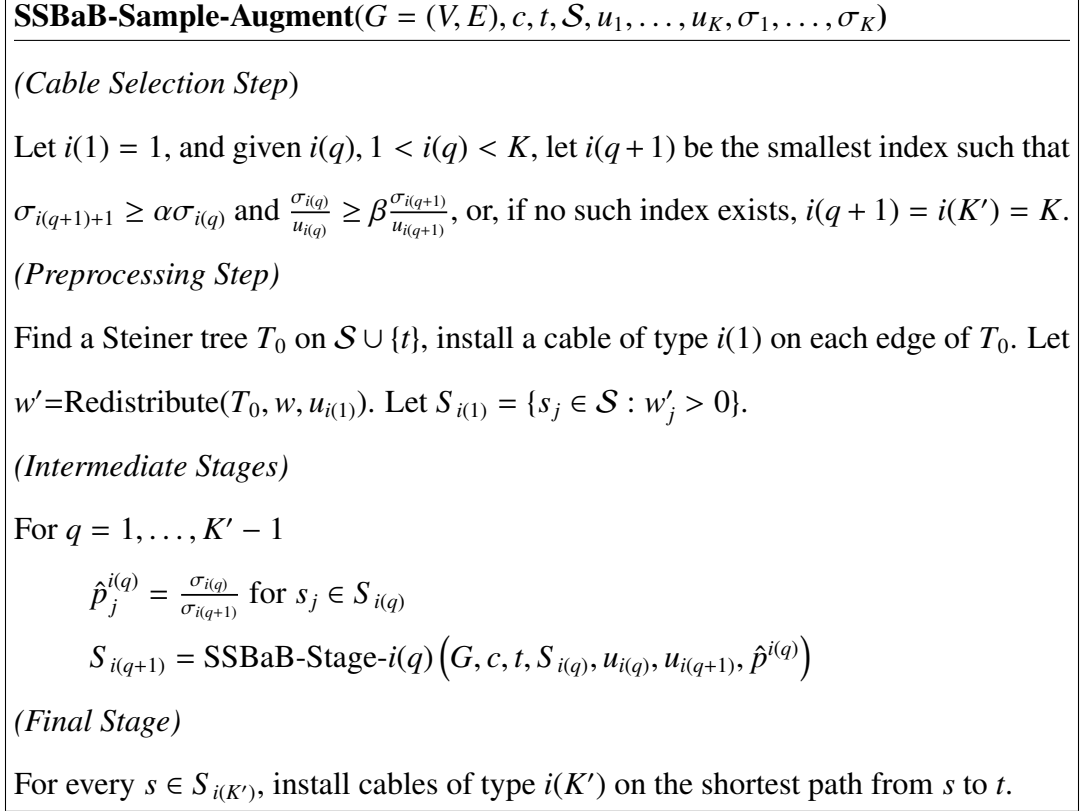


Figure 2.9: Sample-Augment Algorithm for Single-Sink Buy-at-Bulk with Generalized Cable Selection Rule

economies of scale: the cost  $\frac{\sigma_k}{u_k}$  per unit capacity must decrease as the capacity of the cable increases. It turns out that we can drop this assumption; see Remark 4 at the end of this section.

It is not hard to adapt their analysis to our upper bounds, instead of the upper bounds used in Gupta et al. [42]. We give the revised algorithm in Figure 2.9. We replaced the Rounding Step by a Cable Selection Step, which takes two parameters  $\alpha$  and  $\beta$ . We note that in the Intermediate Stages, the algorithm SBBaB-stage- $i(q)$  for  $i(q) = k$  is the same as SSBaB-stage- $k$  given in Figure 2.8, except that we replace  $k+1$  by  $i(q+1)$  (in other words, we only use the cable types  $i(1), \dots, i(K')$ ). If we use  $\alpha = 3.059, \beta = 2.475$  in the algorithm given in Figure 2.9, we get the following result.

**Theorem 22** *There exists a deterministic 27.72-approximation algorithm for the single-sink buy-at-bulk problem.*

**Proof:** We adapt and strengthen our previous analysis; in particular, we slightly change the definitions of  $b^k, r^k(j)$  and  $B_k, R_k(j)$  that we used in the proofs of Lemmas 17 and 20. Since we now have stages  $i(0), \dots, i(K')$ , we define values  $b^{i(q)}, r^{i(q)}(j)$  and  $B_{i(q)}, R_{i(q)}(j)$ . We show that  $b^{i(q)}, r^{i(q)}$  satisfy Lemma 20, and that if  $\hat{p}_j^{i(q)} = \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}}$  for  $s_j \in S_{i(q)}$ , then the total expected cost of the cables installed in stage  $i(q)$  is at most  $B_{i(q)} + \sum_{s_j \in S_{i(q)}} R_{i(q)}(j)$ . This part of the analysis is basically the same as before. The main difference occurs in Lemma 17. We will show that if  $\alpha = 3.059, \beta = 2.475$ , then

$$B_{i(0)} + \sum_{s_j \in S} R_{i(0)}(j) + \sum_{q=1}^{K'} \left( B_{i(q)} + \sum_{s_j \in S} \frac{w_j}{u_{i(q)}} R_{i(q)}(j) \right) \leq 27.72 \text{OPT}.$$

We again define  $b^{i(q)}, r^{i(q)}(j)$  and  $B_{i(q)}, R_{i(q)}(j)$  based on an LP relaxation of the single-sink buy-at-bulk problem. We use a slightly different LP relaxation. Recall that previously, (SSBaB-LP) was defined using the rounded cable capacities and cable costs. We now redefine (SSBaB-LP) on the original parameters, so that we can let the objective value be

$$\min \sum_{e \in E} \sum_{k=1}^K \sigma_k c_e z_e^k.$$

In other words, if we define the LP using the original cable parameters, we do not have to divide the objective value by 4 in order to have (SSBaB-LP) be a relaxation of the single-sink buy-at-bulk problem.

Now, since we assume that  $\frac{\sigma_{k+1}}{u_{k+1}} \leq \frac{\sigma_k}{u_k}$  and  $\sigma_k \leq \sigma_{k+1}$  for all  $k = 1, \dots, K-1$ , it is the case that for every fixed  $q$ ,  $\frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell}$  is increasing in  $\ell$  and  $\frac{\sigma_{i(q+1)}}{\sigma_\ell}$  is decreasing in  $\ell$ . Hence there exists some  $\ell(q)$  such that  $\frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell} \leq \frac{\sigma_{i(q+1)}}{\sigma_\ell}$  for  $\ell \leq \ell(q)$ , and  $\frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell} \geq \frac{\sigma_{i(q+1)}}{\sigma_\ell}$  for

$\ell > \ell(q)$ . Given this definition of  $\ell(q)$  we define

$$r_e^{i(q)}(j) = \sum_{\ell=1}^{\ell(q)} \frac{\hat{x}_e^{j,\ell}}{w_j}, \quad (2.19)$$

$$b_e^{i(q)} = \sum_{\ell=\ell(q)+1}^K \hat{z}_e^\ell, \quad (2.20)$$

where  $\hat{z}, \hat{x}$  is the optimal solution to the SSbAB-LP.

Then it is not hard to show that Lemma 20 holds for stages  $i(1), \dots, i(K' - 1)$ , if we define the values  $B_{i(q)}, R_{i(q)}(j)$  to be

$$B_{i(q)} = \alpha_{i(q)} \sum_{e \in E} \sigma_{i(q+1)} c_e b_e^{i(q)}, \quad R_{i(q)}(j) = \alpha_{i(q)} \sum_{e \in E} \sigma_{i(q)} c_e r_e^{i(q)}.$$

However, instead of setting  $\alpha_{i(q)} = 5$  as we did before, we refine the analysis and set  $\alpha_{i(q)} = 2 \left(1 + \frac{\sigma_{i(q+1)}/u_{i(q+1)}}{\sigma_{i(q)}/u_{i(q)}}\right) \left(1 - \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}}\right) + 2$ . From (2.13) and the arguments following it we see that this setting for  $\alpha_{i(q)}$  rather than 5 is sufficient for Lemma 20 to hold for stage  $i(q)$ .

We refer to the preprocessing step as stage  $i(0)$  and the final stage as  $i(K')$  and we define

$$\begin{aligned} B_{i(0)} &= 2 \sum_{e \in E} \sigma_{i(1)} c_e \sum_{\ell=1}^K \hat{z}_e^\ell b_e^{i(0)}, & R_{i(0)}(j) &= 0, \\ B_{i(K')} &= 0, & R_{i(K')}(j) &= \sum_{e \in E} \sigma_{i(K')} c_e \sum_{\ell=1}^K \frac{\hat{x}_e^{j,\ell}}{w_j}, \end{aligned}$$

which is the same as (2.17) and (2.18). If we let  $\alpha_{i(0)} = 2, \alpha_{i(K')} = 1$  and  $\ell(0) = 0, \ell(K') = K$ , then we can write  $B_{i(q)}, R_{i(q)}$  in a single format for all  $q = 0, \dots, K'$  as

$$\begin{aligned} B_{i(q)} &= \alpha_{i(q)} \sum_{e \in E} \sigma_{i(q+1)} c_e \sum_{\ell=\ell(q)+1}^K \hat{z}_e^\ell, \\ R_{i(q)}(j) &= \alpha_{i(q)} \sum_{e \in E} \sigma_{i(q)} c_e \sum_{\ell=1}^{\ell(q)} \frac{\hat{x}_e^{j,\ell}}{w_j}. \end{aligned}$$

Now, we can follow a similar analysis as in the proof of Lemma 17. First, it follows from the same arguments as in the proof of Lemma 17 that for  $q = 0, \dots, K'$ ,

$$\sum_{s_j \in \mathcal{S}} \frac{w_j}{u_{i(q)}} R_{i(q)}(j) \leq \alpha_{i(q)} \frac{\sigma_{i(q)}}{u_{i(q)}} \sum_{e \in E} c_e \sum_{\ell=1}^{\ell(q)} u_\ell \hat{z}_e^\ell.$$

Therefore

$$\begin{aligned} & B_{i(0)} + \sum_{s_j \in \mathcal{S}} R_{i(0)}(j) + \sum_{q=1}^{K'} \left( B_{i(q)} + \sum_{s_j \in \mathcal{S}} \frac{w_j}{u_{i(q)}} R_{i(q)}(j) \right) \\ & \leq \sum_{q=0}^{K'} \alpha_{i(q)} \sum_{e \in E} c_e \left( \sum_{\ell=1}^{\ell(q)} \sigma_{i(q)} \frac{u_\ell}{u_{i(q)}} \hat{z}_e^\ell + \sum_{\ell=\ell(q)+1}^K \sigma_{i(q+1)} \hat{z}_e^\ell \right) \\ & = \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \left( \sum_{q: \ell(q) < \ell} \alpha_{i(q)} \frac{\sigma_{i(q+1)}}{\sigma_\ell} + \sum_{q: \ell(q) \geq \ell} \alpha_{i(q)} \frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell} \right) \\ & = \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \left( \alpha_{i(0)} \frac{\sigma_{i(1)}}{\sigma_\ell} + \sum_{q=1}^{K'-1} \alpha_{i(q)} \min \left\{ \frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell}, \frac{\sigma_{i(q+1)}}{\sigma_\ell} \right\} + \alpha_{i(K')} \frac{\sigma_{i(K')}/u_{i(K')}}{\sigma_\ell/u_\ell} \right), \end{aligned}$$

where the first equality follows by changing the order of summation, and the second one follows from the definition of  $\ell(q)$ .

Plugging in  $\alpha_{i(0)} = 2, \alpha_{i(K')} = 1$  and  $\alpha_{i(q)} = 2 \left( 1 + \frac{\sigma_{i(q+1)}/u_{i(q+1)}}{\sigma_{i(q)}/u_{i(q)}} \right) \left( 1 - \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}} \right) + 2 = \left( \left( 2 + 2 \frac{\sigma_{i(q+1)}/u_{i(q+1)}}{\sigma_{i(q)}/u_{i(q)}} \right) \left( 1 - \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}} \right) + 2 \right)$  for  $q = 1, \dots, K' - 1$ , and noting that  $\frac{\sigma_{i(K')}/u_{i(K')}}{\sigma_\ell/u_\ell} \leq 1$ , we see that  $\sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell$  is thus charged at most

$$1 + 2 \frac{\sigma_{i(1)}}{\sigma_\ell} + \sum_{q=1}^{K'-1} \left( \left( 2 + 2 \frac{\sigma_{i(q+1)}/u_{i(q+1)}}{\sigma_{i(q)}/u_{i(q)}} \right) \left( 1 - \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}} \right) + 2 \right) \min \left\{ \frac{\sigma_{i(q+1)}}{\sigma_\ell}, \frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell} \right\}$$

times. This expression is the same as the expression  $apx(\ell)$  in Lemma 1 in [39], if we take  $\rho = 2$ :

$$\begin{aligned} apx(\ell) = & 1 + \rho \frac{\sigma_{i(1)}}{\sigma_\ell} + \sum_{q=1}^{K'-1} \left( \left( 2 + 2 \frac{\sigma_{i(q+1)}/u_{i(q+1)}}{\sigma_{i(q)}/u_{i(q)}} \right) \left( 1 - \frac{\sigma_{i(q)}}{\sigma_{i(q+1)}} \right) + \rho \right) \min \left\{ \frac{\sigma_{i(q+1)}}{\sigma_\ell}, \frac{\sigma_{i(q)}/u_{i(q)}}{\sigma_\ell/u_\ell} \right\}. \end{aligned}$$

Now, Grandoni and Italiano show how to bound  $apx(\ell)$  if  $i(1), \dots, i(K')$  are chosen according to the Cable Selection Step, i.e. if  $i(1) = 1$ , and if, given  $i(q)$ ,  $1 < i(q) < K$ ,

$i(q+1)$  is the smallest index such that  $\sigma_{i(q+1)+1} \geq \alpha \sigma_{i(q)}$  and  $\frac{\sigma_{i(q)}}{u_{i(q)}} \geq \beta \frac{\sigma_{i(q+1)}}{u_{i(q+1)}}$ , or, if no such index exists,  $i(q+1) = i(K') = K$ . Equations (7-12) in the proof of Theorem 2 in [39] state that if  $\alpha > 1, \beta > 1$  and  $\beta \leq 3 + \frac{\rho}{2}$  then

$$apx(\ell) \leq \max \left\{ \begin{array}{l} 1 + (2 + \frac{2}{\beta} + \rho)(1 + \beta + \frac{\beta}{\beta-1}) + \rho \frac{2}{\alpha-1}, \\ 1 + (2 + \frac{2}{\beta} + \rho)(\alpha + \frac{\beta}{\beta-1}) + \rho \frac{\alpha+1}{\alpha-1}, \\ 1 + (2 + \frac{2}{\beta} + \rho)(1 + \frac{\beta}{\beta-1}) + \rho \frac{\alpha+1}{\alpha-1}, \\ 1 + (2 + \frac{2}{\beta} + \rho) + \rho \frac{2}{\alpha-1} + (4 + \rho)\beta, \\ 1 + \rho \frac{\alpha+1}{\alpha-1} + (4 + \rho)\alpha, \\ 1 + \rho \frac{\alpha+1}{\alpha-1} + (4 + \rho) \end{array} \right\}.$$

Hence if we set  $\alpha = 3.059, \beta = 2.475$ , then  $apx(\ell)_{[\rho=2]} \leq 27.72$  for  $\ell = 1, \dots, K$ .

Hence we have shown that

$$B_{i(0)} + \sum_{s_j \in S} R_{i(0)}(j) + \sum_{q=1}^{K'} \left( B_{i(q)} + \sum_{s_j \in S} \frac{w_j}{u_{i(q)}} R_{i(q)}(j) \right) \leq 27.72 \sum_{\ell=1}^K \sum_{e \in E} \sigma_\ell c_e \hat{z}_e^\ell \leq 27.72 OPT.$$

The arguments leading up to Corollary 21 show that these observations are enough to show that there exists a deterministic 27.72-approximation algorithm for the single-sink rent-or-buy problem.  $\square$

**Remark 4** We note that for our analysis, the assumption that the cables satisfy economies of scale is without loss of generality, because we use the optimal value of SSBaB-LP as an upper bound: if  $\frac{\sigma_{k+1}}{u_{k+1}} > \frac{\sigma_k}{u_k}$ , then the optimal solution to SSBaB-LP will not use cable type  $k+1$ , since it can instead use  $\frac{u_{k+1}}{u_k}$  cables of type  $k$  which results in a lower cost. Grandoni and Italiano [39] use the (integer) optimum as an upper bound, and since  $\frac{u_{k+1}}{u_k}$  may not be integer, they do need the additional assumption of economies of scale.

## 2.4 Conclusion

We propose a specific method for derandomizing Sample-Augment algorithms, and we successfully apply this method to all but one of the Sample-Augment algorithms in Gupta et al. [42], and to the *a priori* traveling salesman problem and the 2-stage rooted stochastic Steiner tree problem with independent decisions.

The question whether the Sample-Augment algorithm for multicommodity rent-or-buy problem can be derandomized remains open. The multicommodity rent-or-buy problem is a generalization of the single source rent-or-buy problem: instead of one source  $s$  and sinks  $t_1, \dots, t_k$ , we are given  $k$  source-sink pairs  $(s_1, t_1), \dots, (s_k, t_k)$  and need to construct a network so that each source-sink pair is connected. The Sample-Augment algorithm for this problem [42, 30] marks each source-sink pair with probability  $\frac{1}{M}$  and buys a Steiner forest on the marked terminals in the Subproblem Step. In the Augmentation Step, we contract the bought edges, and rent the shortest path connecting each terminal pair in the contracted graph. If we want to use Theorem 1, we would need to be able to compute  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  (or a good upper bound for it) efficiently and it is unclear how to do this for the multicommodity rent-or-buy algorithm, because unlike in the algorithms we discussed,  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$  does depend on the subproblem solution, and not just on  $\mathbb{D}$ .

It may also be possible to extend our approach to the Boosted Sampling algorithms for two-stage stochastic optimization problems [44], especially for the special case of independent decisions, but except for the rooted Steiner tree problem it is not obvious how to determine  $\mathbb{E}_p[C_{aug}(\mathbb{D})]$ . There is a similar but even larger obstacle if we want to use our techniques to derandomize the Boosted Sampling algorithms for multi-stage stochastic optimization problems, because here we would also need to be able to com-



pute (an upper bound on) the expected cost of future stages.

# CHAPTER 3

## DETERMINISTIC PIVOTING ALGORITHMS FOR RANKING AND CLUSTERING

### 3.1 Introduction

In this chapter, we consider the problem of ranking or clustering a set of elements, based on input information for each pair of elements. The objective is to find a solution that minimizes the deviation from the input information.

Consider, for example, the problem of ranking the teams of a round-robin sports tournament: each pair of teams has played against each other exactly once, and a desirable property of a ranking of the teams is that it ranks team  $i$  before team  $j$  if team  $i$  won the game against team  $j$ . Clearly, this is not always possible, since there may be teams  $i, j, k$  where team  $i$  beat team  $j$ , team  $j$  beat team  $k$  and team  $k$  beat team  $i$ . An example of clustering that has a similar flavor to the round-robin sports tournament is clustering web pages based on similarity scores. For each pair of pages we have a score between 0 and 1, and we would like to cluster the pages so that pages with a high similarity score are in the same cluster, and pages with a low similarity score are in different clusters. We will also consider hierarchical clustering. Given a set of elements and integer values  $D_{ij}$  between 0 and  $M$  for each pair of elements  $i, j$ , we want to find a hierarchical clustering of  $M$  levels, i.e.  $M$  nested partitions of the elements, such that elements  $i, j$  are in different clusters at  $D_{ij}$  levels. As in the sports tournament example, not every input admits a solution that exactly satisfies these desired properties. We will therefore seek a solution that minimizes the sum of the pairwise deviations from the input information.

Other examples arise in aggregation of existing rankings or clusterings. For exam-

ple, Dwork, Kumar, Naor and Sivakumar [19] propose meta-search engines for Web search, where we want to get more robust rankings that are not sensitive to the various shortcomings and biases of individual search engines by combining the rankings of individual search engines. We will be considering this application in more detail in Chapter 4. Another example comes from gene expression analysis. The goal is to find classifications of genes based on the results from different microarray experiments. In this context, Filkov and Skiena [29] introduce the problem called consensus clustering, in which we want to find a clustering of a set of elements that is as close as possible to a collection of clusterings of the same set of elements.

These examples are related to the topic of voting systems, in which each voter gives a preference on a set of alternatives, and the system outputs a single preference order on the set of alternatives based on the voters' preferences. Arrow's impossibility theorem [7] states that no voting system can simultaneously achieve non-dictatorship, independence of irrelevant alternatives, and Pareto efficiency. Non-dictatorship means that the voting system does not simply output the preference order of a particular voter, independence of irrelevant alternatives means that if the output order ranks  $i$  before  $j$ , then this should also be the case if one or more voters change the position of an alternative  $k \neq i, j$ , and finally, Pareto efficiency or unanimity ensures that if all voters prefer alternative  $i$  over alternative  $j$ , then so does the output ordering. Kemeny [52] proposed the following objective for determining the best voting system: Given permutations  $\pi_1, \dots, \pi_\ell$  of  $V$ , we want to find  $\pi$  that minimizes  $\sum_{k=1}^{\ell} \mathcal{K}(\pi, \pi_k)$ , where  $\mathcal{K}(\pi, \pi_k)$  is the Kendall tau distance, which is defined as the number of pairs  $i, j$  such that  $\pi_k(i) < \pi_k(j)$  and  $\pi(i) > \pi(j)$ . In other words, in Kemeny aggregation we want to find a permutation that minimizes the number of pairwise disagreements with the  $\ell$  input permutations.

Fagin, Kumar, Mahdian, Sivakumar and Vee [25] extend the problem to *partial* rank

aggregation (see also Ailon [2]). The input rankings do not have to be permutations of the same set of elements, but are instead allowed to have ties. More precisely, a partial ranking of a set of elements  $V$  is a function  $\pi : V \rightarrow \{1, \dots, |V|\}$ . Examples of partial rankings are top- $m$  rankings, i.e. permutations of only a subset of the elements (in which case we make the assumption that the unranked elements all share the position after the last ranked element), or the rankings may be  $p$ -ratings, i.e. mappings from  $V$  to  $\{1, \dots, p\}$ , as is the case in movie rankings. Fagin et al. propose several metrics to compare partial rankings, and show that they are within constant multiples of each other. Following Ailon, we will say the distance  $\mathcal{K}(\pi_1, \pi_2)$  between two partial rankings  $\pi_1$  and  $\pi_2$  is again the number of pairs  $i, j$  such that  $\pi_1(i) < \pi_1(j)$ , and  $\pi_2(i) > \pi_2(j)$ . The goal of partial rank aggregation is given partial rankings  $\pi_1, \dots, \pi_\ell$  to find a *permutation* that minimizes  $\sum_{k=1}^{\ell} \mathcal{K}(\pi, \pi_k)$ . Note that the output is required to be a permutation, and cannot be a partial ranking. We will refer to Kemeny aggregation and partial rank aggregation simply as rank aggregation. If the input rankings are permutations rather than partial rankings we will use the term full rank aggregation.

Consensus clustering as defined by Filkov and Skiena [29] is similar to rank aggregation. Given  $\ell$  clusterings (partitions)  $C_1, \dots, C_\ell$  of a set  $V$ , we want to find a clustering  $C$  that minimizes  $\sum_{k=1}^{\ell} d(C_k, C)$ , where  $d(C_k, C)$  is the number of pairs  $i, j$  such that  $i, j$  are in the same cluster in one of the clusterings  $C_k, C$ , and in different clusters in the other clustering. We introduce here the problem of *partial* consensus clustering, which is similar to partial rank aggregation. The input is  $C_1, \dots, C_\ell$  where each  $C_k$  is a collection of disjoint subsets of  $V$ . Unlike in regular consensus clustering, we no longer require that  $\bigcup_{C \in C_k} C = V$ . We will call such a collection  $C_k$  a partial clustering of  $V$ . We define the distance  $d(C, \mathcal{D})$  between two (partial) clusterings  $C$  and  $\mathcal{D}$  as the number of pairs  $i, j$  such that one of the two collections  $C, \mathcal{D}$  contains a set that contains both  $i, j$ , and the other collection does not. For example, if  $V = \{1, 2, 3, 4\}$ , and

$C = \{\{1, 2\}\}, \mathcal{D} = \{\{1, 3\}, \{2, 4\}\}$ , then  $d(C, \mathcal{D}) = 3$ , since the pairs contributing to the distance are  $\{1, 2\}, \{1, 3\}$  and  $\{2, 4\}$ . If  $C, \mathcal{D}$  are full clusterings or partitions of  $V$  then the distance function is the same as the one defined previously. As before the goal of partial consensus clustering is to find a clustering (partition)  $C$  that minimizes  $\sum_{k=1}^{\ell} d(C_k, C)$ . From now on, we will refer to the problem when the input clusterings are partitions of  $V$  as full consensus clustering.

We believe that partial consensus clustering is a useful extension of consensus clustering, because it seems more appropriate when we want to cluster a dataset based on different features. As an example we mention a problem studied by Modha and Spangler [57, 56] of clustering web search results based on three criteria: words contained in the document, out-links from the document, and in-links to the document. Documents are assumed to be similar if they share one or more words, and if they share one or more in-links or out-links. Although this is not the approach proposed in [57, 56], a possible approach to this is to find a clustering based on each of the three criteria, and aggregate the three resulting clusterings using consensus clustering. But what if we have some pages that have no out-links? Such a document is not similar to any other document based on the out-link criterion, since it does not share an out-link with any document. Hence we could say it is dissimilar to any document that does have out-links, but it really is neither similar nor dissimilar to another document that does not have out-links. A partial clustering allows us to represent this information by clustering only the documents that do have out-links, and using this clustering as one of the input clusterings to the consensus clustering algorithm.

We now give a general framework for the ranking problems and the clustering problems that we consider in this chapter. In the *weighted minimum feedback arc set problem*, we are given a set of vertices  $V$ , nonnegative weights  $w = \{w_{(i,j)}, w_{(j,i)} : i \in V, j \in$

$V, i \neq j\}$ , where we assume without loss of generality that the weights are scaled so that  $w_{(i,j)} + w_{(j,i)} \leq 1$ . We want to find a permutation  $\pi$  that minimizes the weight of pairs of vertices out of order with respect to the permutation, i.e.  $\sum_{\pi(i) < \pi(j)} w_{(j,i)}$ . The (unweighted) feedback arc set problem has weights that are either 0 or 1. The best known approximation algorithm for this problem has a performance guarantee of  $O(\log n \log \log n)$  [61, 24], and it has the same hardness of approximation as vertex cover [51]. In this thesis, we will consider weighted feedback arc set problems where the weights satisfy one of the following: *probability constraints*: for any pair  $i, j$ ,  $w_{(i,j)} + w_{(j,i)} = 1$ , or the *triangle inequality*: for any triple  $i, j, k$ ,  $w_{(i,j)} + w_{(j,k)} \geq w_{(i,k)}$ . We will sometimes refer to these problems as the *ranking* problem. The feedback arc set problem in tournaments is the special case when the weights satisfy the probability constraints and are either 0 or 1. Note that we can represent the problem of ranking the teams of a round-robin sports tournament as a feedback arc set in tournaments by setting  $w_{(i,j)} = 1$  if team  $i$  beat team  $j$ . Rank aggregation is also a special case of weighted minimum feedback arc set, since we can set  $w_{(i,j)} = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbf{1}\{\pi_k(i) < \pi_k(j)\}$  where  $\mathbf{1}\{\cdot\}$  is the indicator function. Note that these weights satisfy the triangle inequality, and in the case of full rank aggregation also the probability constraints.

In the *weighted clustering problem*, we are given a set of vertices  $V$  and nonnegative weights  $w^+ = \{w_{\{i,j\}}^+ : i \in V, j \in V, i \neq j\}$ ,  $w^- = \{w_{\{i,j\}}^- : i \in V, j \in V, i \neq j\}$  satisfying  $w_{\{i,j\}}^+ + w_{\{i,j\}}^- \leq 1$  for every  $i, j$ . We want to find a clustering  $C$  minimizing  $\sum_{\{i,j\} \in C} w_{\{i,j\}}^+ + \sum_{\{i,j\} \notin C} w_{\{i,j\}}^-$ . The unweighted clustering problem in which each weight is either 0 or 1 is the correlation clustering problem (in general graphs). The best known approximation algorithm for this problem has a performance guarantee of  $O(\log n)$  [14]. In this thesis, we consider two special cases. We say the weights satisfy *probability constraints* if for every  $i, j \in V$ ,  $w_{\{i,j\}}^+ + w_{\{i,j\}}^- = 1$ . We will say the weights satisfy the *triangle inequality* if for every triple  $i, j, k$ ,  $w_{\{i,j\}}^- + w_{\{j,k\}}^- \geq w_{\{i,k\}}^-$  and

$w_{\{i,j\}}^+ + w_{\{j,k\}}^- \geq w_{\{i,k\}}^+$ . We note that the triangle inequality assumption in [4] only assumes the first set of inequalities. The problem where exactly one of  $w_{\{i,j\}}^+$  and  $w_{\{i,j\}}^-$  is 1 (and the other 0) is known as *correlation clustering* (in complete graphs). We will use the term correlation clustering to refer to correlation clustering in complete graphs. Given an instance  $C_1, \dots, C_\ell$  of consensus clustering, we can define  $w_{\{i,j\}}^+ = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbf{1}\{\exists C \in C_k \text{ s.t. } \{i, j\} \subset C\}$ , and  $w_{\{i,j\}}^- = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbf{1}\{\exists C \in C_k \text{ s.t. exactly one of } i, j \text{ is in } C\}$ . It is easily verified that the weights satisfy the triangle inequality. If the input clusterings are partitions, the weights also satisfy the probability constraints.

*Hierarchical clustering* can be seen as a generalization of correlation clustering. An  $M$ -level hierarchical clustering of a set  $V$  is a nested clustering of the vertices in  $V$ , where the clustering at level  $m$  is a refinement of the clustering at level  $m + 1$ . Given a set  $V$  and a matrix  $D$  with  $D_{ij} \in \{0, \dots, M\}$  for any distinct  $i, j \in V$ , we want to find an  $M$ -level hierarchical clustering of  $V$  minimizing  $\sum_{i,j \in V} |D_{ij} - \lambda_{ij}|$ , where  $\lambda_{ij}$  is the number of levels in which  $i$  and  $j$  are in different clusters, or equivalently, since the clusterings are nested,  $i$  and  $j$  are in different clusters at levels  $1, \dots, \lambda_{ij}$ , and in the same cluster at levels  $\lambda_{ij} + 1, \dots, M$ . It was shown in [47] that this is equivalent to finding an *ultrametric* that minimizes the  $\ell_1$  distance to  $D$ . An ultrametric is a tree metric in which all vertices are at the leaves of the tree, and the distance from each leaf to the root is the same.

We also consider *constrained* versions of these problems, which were introduced by Hegde and Jain [49] (see also [66]). The constraints we consider are constraints on pairs of vertices. In the case of clustering, constraints can be given as sets  $P^+, P^- \in V \times V$  and any feasible clustering must have a pair  $i, j$  in the same cluster if  $\{i, j\} \in P^+$ , and in different clusters if  $\{i, j\} \in P^-$ . In hierarchical clustering, in addition to matrix  $D$ , we are given matrices  $L, U$  such that  $L_{ij} \leq D_{ij} \leq U_{ij}$  for every  $i, j \in V$ . Any feasible hierarchical clustering must have  $L_{ij} \leq \lambda_{ij} \leq U_{ij}$ , where  $\lambda_{ij}$  is again the number of levels in which  $i$

and  $j$  are in different clusters. In the ranking case, the constraints are a partial order  $P$ , and any feasible solution must be a linear extension of  $P$ . These constraints can reflect prior beliefs about the output ranking or clustering; for example, when aggregating the ranked results of different search engines, we may want to ensure that the top-level page of a web site is ordered before subpages of the site, or when clustering web pages we can ensure all pages of a web site are in the same cluster.

### 3.1.1 Related Work

The feedback arc set problem in tournaments is NP-hard [4, 5, 13]. Rank aggregation is also NP-hard [10], even if the number of input rankings is only 4 [19]. Correlation clustering is MAX-SNP hard [14], and consensus clustering is NP-hard [67, 54], although it is not known to be NP-hard if the number of input clusterings is constant.

The maximization versions of these problems, where instead of minimizing the pairwise deviations from the input information we try to maximize the pairwise agreements, are “easy” to approximate: there exist polynomial time approximation schemes (PTAS) for the corresponding maximization problems of feedback arc set in tournaments [6, 31] and correlation clustering [9].

Ailon, Charikar and Newman [4] give the first constant-factor approximation algorithms for the unconstrained ranking and clustering problems with weights that satisfy either triangle inequality constraints, probability constraints, or both. Their algorithms are randomized and based on Quicksort: the algorithms recursively generate a solution by choosing a random vertex as a “pivot” and ordering all other vertices with respect to the pivot vertex according to some criterion. In the first type of algorithm they give for the ranking problem, a vertex  $j$  is ordered before the pivot  $k$  if  $w_{(j,k)} \geq w_{(k,j)}$  or or-



dered after  $k$  otherwise. Next, the algorithm recurses on the two instances induced by the vertices before and after the pivot. In the case of a clustering problem, a vertex  $j$  is placed in the same cluster as the pivot vertex  $k$  if  $w_{\{j,k\}}^+ \geq w_{\{j,k\}}^-$ . The algorithm recurses on the instance induced by the vertices that are not placed in the same cluster as the pivot vertex. The second type of algorithm first solves a linear programming relaxation of the problem under consideration, which has variables  $x_{(i,j)}$  and  $x_{(j,i)} = 1 - x_{(i,j)}$  or  $x_{\{i,j\}}^+$  and  $x_{\{i,j\}}^- = 1 - x_{\{i,j\}}^+$  for every pair  $i, j \in V$  for the ranking and clustering problems respectively. The pivoting scheme is then used to randomly round the fractional solution, i.e. if  $k$  is the pivot vertex, then  $j$  is ordered before  $k$  (clustered together with  $k$ ) with probability  $x_{(j,k)}$  ( $x_{\{j,k\}}^+$ ) and ordered after  $k$  (not clustered in the same cluster as  $k$ ) with probability  $x_{(k,j)}$  ( $x_{\{j,k\}}^-$ ).

In the case of rank aggregation and consensus clustering, a folklore result is that returning the best of the input rankings or clusterings is a 2-approximation algorithm. Ailon et al. also show that one can obtain better approximation factors for rank aggregation and consensus clustering by returning the best of their algorithm's solution and the best input ranking/clustering. For instance, for rank aggregation, they obtain a randomized  $\frac{11}{7}$ -approximation algorithm using their first type of algorithm, and a randomized  $\frac{4}{3}$ -approximation algorithm using their second, LP-based, algorithm.

There has been a good deal of follow-up work since the Ailon et al. paper. Ailon and Charikar [3] consider hierarchical clustering and fitting ultrametrics. An ultrametric satisfies the property that for any three distinct vertices, the two largest pairwise distances are equal. An  $M$ -level hierarchical clustering is a special case of an ultrametric in which the distance between each pair of vertices is an integer between 0 and  $M$ . Their algorithm for hierarchical clustering takes a random pivot vertex  $k$ , and adjusts the distance of a pair  $i, j$  if the triple  $i, j, k$  does not satisfy the ultrametric property. They show that

this gives an expected  $(M + 2)$ -approximation algorithm. They also give an LP-based  $O(\log n \log \log n)$  approximation algorithm for fitting ultrametrics.

Coppersmith, Fleischer, and Rudra [17] give a simple, greedy 5-approximation algorithm for the ranking problem when weights obey the probability constraints. Kenyon-Mathieu and Schudy [53] give a polynomial-time approximation scheme for unconstrained weighted feedback arc set in tournaments with weights satisfying  $b \leq w_{(i,j)} + w_{(j,i)} \leq 1$  for all  $i, j \in V$  for some  $b > 0$ . Note that this includes problems satisfying the probability constraints and hence includes the full rank aggregation problem as a special case. Their approximation scheme assumes the availability of a solution with cost that is not more than a constant factor  $\alpha$  from optimal. To get a  $(1 + \epsilon)$ -approximate solution, the running time of their algorithm is doubly exponential in  $\frac{1}{\epsilon}$ ,  $\frac{1}{b}$  and  $\alpha$ .

Ailon [2] considers the partial rank aggregation problem. He generalizes and improves some of the results from Ailon et al. to partial rank aggregation. He shows that perturbing the solution to the linear programming relaxation and using these perturbed values as probabilities gives a randomized  $\frac{3}{2}$ -approximation algorithm for partial rank aggregation. Since his analysis only uses the fact that the weights satisfy the triangle inequality, this also yields  $\frac{3}{2}$ -approximation algorithm for ranking with triangle inequality constraints on the weights.

### 3.1.2 Our Results

We show how to give deterministic algorithms matching the best randomized algorithms of [4], [2], and [3]. This answers an open question in these works. The techniques from Ailon et al. are not amenable to standard techniques of derandomization, but we show that we can amortize in place of the expectation and make the randomized algorithm

deterministic.

In Section 3.2 and Section 3.3 respectively, we give very simple deterministic algorithms as derandomizations of the combinatorial algorithms in [4]. The analysis of these algorithms is simpler than the analysis of the original algorithms, and actually implies the performance guarantees for the original algorithms as well.

In Section 3.4 we consider constrained problems and show that any approximation algorithm for rank aggregation and consensus clustering also implies the same guarantee for constrained versions of the problem. For weighted feedback arc set and clustering with probability constraints (but not triangle inequality) we show that our algorithms from Section 3.2 and 3.3 can also deal with constrained problems.

In Section 3.5, we extend the ideas from Sections 3.2 and 3.3 to the randomized rounding algorithms in [4] and [2], and show how to derandomize these algorithms. Although the analysis of these algorithms gets more involved, our analysis here is again simpler than the analysis of the original randomized algorithms, and again implies the approximation guarantees of the original randomized algorithms.

Finally, in Section 3.6, we show how to use the algorithm in Section 3.3 to obtain a deterministic  $(M + 2)$ -approximation algorithm for  $M$ -level hierarchical clustering. The algorithm and its analysis follow easily from Section 3.3, in contrast to the (expected)  $(M + 2)$ -approximation algorithm in [3], which is rather involved.

## 3.2 A Simple Ranking Algorithm

Ailon, et al. [4] propose a simple algorithm to obtain a permutation that costs at most 2 times the optimum if the weights satisfy the triangle inequality, or 3 times the optimum

<b>FAS-Pivot(<math>G = (V, A)</math>)</b>
Pick a pivot $k \in V$ . $V_L = \{i \in V : (i, k) \in A\}$ , $V_R = \{i \in V : (k, i) \in A\}$ . Return FAS-Pivot( $G(V_L)$ ), $k$ , FAS-Pivot( $G(V_R)$ ).

Figure 3.1: FAS-Pivot Algorithm

in the case of probability constraints. Given an instance of the weighted feedback arc set problem, they form a tournament  $G = (V, A)$  by including arc  $(i, j)$  only if  $w_{(i,j)} \geq w_{(j,i)}$  (breaking ties arbitrarily). This is called the majority tournament [4]. Note that if the majority tournament is acyclic, it corresponds to an optimal permutation: the cost for pair  $i, j$  in any solution is at least  $\min\{w_{(i,j)}, w_{(j,i)}\}$ , and this lower bound is met for every pair. We give a general framework, FAS-Pivot in Figure 3.1, which generalizes both our algorithm, and the algorithm in [4]. We use the following notation: We denote by  $G(V') = (V', A(V'))$  the subgraph of  $G$  induced by  $V' \subseteq V$ . If  $\pi_1$  and  $\pi_2$  are permutations of disjoint sets  $V_1, V_2$ , we let  $\pi_1, \pi_2$  denote the concatenation of the two permutations. In Ailon et al.’s algorithm, a pivot is chosen randomly. In our deterministic version of this algorithm, we will use a lower bound on the weight of an optimal solution to give a way of always choosing a “good” pivot. We will first state our algorithm, and in particular our choice of pivot, in general terms, and we then show how this implies a 2-approximation algorithm for weighted feedback arc set with triangle inequality, a  $\frac{8}{5}$ -approximation algorithm for partial rank aggregation, and a 3-approximation algorithm for weighted feedback arc set with probability constraints.

Our algorithms will have a “budget”  $c_{ij}$  for every pair of vertices  $i, j$ . The budgets will be chosen in such a way that the total budget is a lower bound on the value of the

optimal solution. Theorem 23 below gives conditions under which we can show that on average the cost incurred by a vertex pair is not more than  $\alpha$  times its budget.

The first condition of Theorem 23 states that the cost of ordering  $i$  before  $j$  if  $(i, j) \in A$  is at most  $\alpha$  times its budget. Note that the only way a pair of vertices  $i, j$  is *not* ordered according to  $A$  (i.e.  $j$  is ordered before  $i$  even though  $(i, j) \in A$ ), is if in some recursive call  $j$  ends up in  $V_L$ , and  $i$  in  $V_R$ . For a pivot  $k$ , let  $T_k(G)$  be the set of arcs for which this occurs, if  $k$  is the pivot and the input to the recursive call is  $G = (V, A)$ , i.e.  $T_k(G) = \{(i, j) \in A : (j, k) \in A, (k, i) \in A\}$ . Our algorithm chooses the pivot that minimizes the ratio of the cost for these arcs to their budget.

We now prove the following key theorem, which states conditions under which FAS-Pivot can be used to obtain a solution to a given input of a weighted feedback arc set problem, that costs at most  $\alpha$  times a given budget  $\sum_{i \in V, j \in V, i < j} c_{ij}$ .

**Theorem 23** *Given an input  $(V, w)$  of the weighted feedback arc set problem, a set of budgets  $\{c_{ij} : i \in V, j \in V, i \neq j\}$ , and a tournament  $G = (V, A)$  such that*

$$(i) \ w_{(j,i)} \leq \alpha c_{ij} \text{ for all } (i, j) \in A,$$

$$(ii) \ w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq \alpha(c_{ij} + c_{jk} + c_{ki}) \text{ for any directed triangle } (i, j), (j, k), (k, i) \text{ in } A,$$

*then FAS-Pivot returns a solution that costs at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$  if we choose a pivot  $k$  that minimizes*

$$\frac{\sum_{(i,j) \in T_k(G)} w_{(i,j)}}{\sum_{(i,j) \in T_k(G)} c_{ij}}. \quad (3.1)$$

---

<sup>1</sup> Throughout this chapter, we define a ratio to be 0 if both numerator and denominator are 0. If only the denominator is 0, we define it to be  $\infty$ .

**Proof:** Let  $G = (V, A)$  be a tournament that satisfies the conditions in the theorem. For a pair of vertices  $i, j$  with  $(i, j) \in A$ , we will say  $i, j$  incurs a “forward” cost if  $i$  is ordered to the left of  $j$ , and we say  $i, j$  incurs a “backward” cost if  $i$  is ordered to the right of  $j$ . In the latter case, we will also say that  $(i, j)$  becomes a backward arc.

Let an “iteration” be the work done by  $\text{FAS-Pivot}(G)$  before the recursive call. We bound the forward and backward costs that are incurred in one iteration by  $\alpha$  times the respective budgets of the vertex pairs involved. Since the conditions of the theorem will also hold in subsequent iterations, and since each vertex pair incurs either a forward or a backward cost in exactly one iteration, this then guarantees that we find a solution of at most  $\alpha$  times the total budget given by  $\sum_{i \in V, j \in V, i < j} c_{ij}$ .

Note that the first condition implies that a forward cost incurred by a pair  $i, j$  is at most  $\alpha c_{ij}$ . We now show that the second condition implies that we always choose a pivot so that the total backward cost incurred by pivoting on this vertex is at most  $\alpha$  times the budget for these vertex pairs.

For a given pivot  $k$ ,  $T_k(G) \subset A$  is the set of arcs that become backward by pivoting on  $k$ . The backward cost incurred if  $k$  is the pivot is equal to  $\sum_{(i,j) \in T_k(G)} w_{(i,j)}$ , and we have a budget for these vertex pairs of  $\sum_{(i,j) \in T_k(G)} c_{ij}$ . Hence we need to show that we always choose a pivot such that the ratio in (3.1) is at most  $\alpha$ . Note that this can be done by showing that

$$\sum_{k \in V} \sum_{(i,j) \in T_k(G)} w_{(i,j)} \leq \alpha \sum_{k \in V} \sum_{(i,j) \in T_k(G)} c_{ij}. \quad (3.2)$$

We observe that  $(i, j)$  becomes a backward arc if  $(k, i)$  and  $(j, k)$  in  $A$ , in other words, exactly when  $(i, j)$  is in a directed triangle with the pivot  $k$ . Therefore  $T_k(G)$  contains exactly the arcs that are in a directed triangle with  $k$  in  $G$ . Let  $T$  be the set of directed triangles in  $G$ , and for a triangle  $t = \{(i, j), (j, k), (k, i)\}$ , let  $w(t) = \sum_{(g,h) \in t} w_{(g,h)}$  and let

$c(t) = \sum_{(g,h) \in t} c_{gh}$ . If we sum  $\sum_{(i,j) \in T_k(G)} w_{(i,j)}$  over all  $k \in V$ , i.e.  $\sum_{k \in V} \sum_{(i,j) \in T_k(G)} w_{(i,j)}$ , then we count  $w_{(i,j)}$  exactly once for every pivot  $k$  such that  $(i, j), (j, k), (k, i)$  is a directed triangle, hence  $\sum_{k \in V} \sum_{(i,j) \in T_k(G)} w_{(i,j)} = \sum_{t \in T} w(t)$ . Similarly,  $\sum_{(i,j) \in T_k(G)} c_{ij} = \sum_{t \in T} c(t)$ .

By the second condition of the theorem,  $w(t) \leq \alpha c(t)$ , therefore (3.2) holds, and hence there must exist some pivot  $k$  such that  $\sum_{(i,j) \in T_k(G)} w_{(i,j)} \leq \alpha \sum_{(i,j) \in T_k(G)} c_{ij}$ .  $\square$

**Remark 5** *The proof of Theorem 23 also implies that under the conditions of the theorem, choosing a pivot at random gives a solution with expected cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$ .*

**Lemma 24** *The algorithm in Theorem 23 can be implemented in  $O(n^3)$  time.*

**Proof:** We maintain a list of the directed triangles in  $G$  for which all three vertices are currently contained in a single recursive call, and for each vertex we maintain the total cost for the arcs that become backward if pivoting on that vertex and the total budget for these pairs (i.e. the numerator and denominator of (3.1)).

If we disregard the time needed to obtain and update this information, then a single recursive call takes  $O(n)$  time, and there are at most  $O(n)$  iterations, giving a total of  $O(n^2)$ .

Initializing the list of triangles and the numerator and denominator of (3.1) for each vertex takes  $O(n^3)$  time. Over all recursive calls combined, the time needed to update the list of directed triangles, and the numerator and denominator of (3.1) is  $O(n^3)$ : After each pivot, we need to remove all triangles that either contain the pivot vertex, or contain  $(i, j)$  where  $i$  and  $j$  are separated into different recursive calls, and for each triangle removed from the list, we need to update the numerator and denominator of (3.1) for the three vertices in the triangle. Assuming the list of triangles is linked to the vertices and arcs

contained in it and vice versa, finding a triangle that contains a certain vertex or arc, removing it, and updating the numerator and denominator for the vertices contained in it, can be done in constant time. Finally, note that each triangle is removed from the list exactly once, hence the overall time to update the list of directed triangles, and the numerator and denominator of (3.1) is  $O(n^3)$ .  $\square$

### 3.2.1 Weighted Feedback Arc Set with Triangle Inequality

We now show that Theorem 23 implies a 2-approximation algorithm for weighted feedback arc set with triangle inequality. We then use the theorem plus a 2-approximation algorithm from Ailon [2] to obtain an improved performance guarantee for rank aggregation problems, where in addition to the weights we have the original input rankings.

**Theorem 25** *There exists a deterministic, combinatorial 2-approximation algorithm for weighted feedback arc set in tournaments with triangle inequality which runs in  $O(n^3)$  time.*

**Proof:** Let  $G = (V, A)$  be the majority tournament, i.e.  $(i, j) \in A$  only if  $w_{(i,j)} \geq w_{(j,i)}$  (breaking ties arbitrarily). Let  $c_{ij} = \min\{w_{(i,j)}, w_{(j,i)}\}$  for every  $i \in V, j \in V$ . Note that  $\sum_{i \in V, j \in V, i < j} c_{ij}$  is a lower bound on the weight of any feasible solution. We will show that the conditions of Theorem 23 are satisfied with  $\alpha = 2$ , hence Theorem 23 plus Lemma 24 imply the result.

The first condition is met with  $\alpha \geq 1$ , since if  $(i, j) \in A$ , then  $w_{(j,i)} \leq w_{(i,j)}$ , hence  $w_{(j,i)} = \min\{w_{(i,j)}, w_{(j,i)}\} = c_{ij}$ .



Let  $t = \{(i, j), (j, k), (k, i)\}$  be a directed triangle in  $G$ , then by the triangle inequality on the weights,  $w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq (w_{(i,k)} + w_{(k,j)}) + (w_{(j,i)} + w_{(i,k)}) + (w_{(k,j)} + w_{(j,i)}) = 2(w_{(j,i)} + w_{(k,j)} + w_{(i,k)})$ . Now note that  $(i, j) \in A$  implies that  $c_{ij} = w_{(j,i)}$  and similarly,  $c_{jk} = w_{(k,j)}, c_{ki} = w_{(i,k)}$ . Hence  $w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq 2(c_{ij} + c_{jk} + c_{ki})$ , and the second condition is satisfied for  $\alpha = 2$ .  $\square$

### 3.2.2 Rank Aggregation

As in Ailon et al. [4], we can do better in the case of rank aggregation. In fact, we will extend the ideas from Ailon et al. [4], and Ailon [2] to give a combinatorial  $\frac{8}{5}$ -approximation algorithm for partial rank aggregation.

Recall that in the partial rank aggregation problem, we have  $\ell$  partial rankings  $\pi_1, \dots, \pi_\ell$  such that  $w_{(i,j)} = \frac{1}{\ell} \sum_{\ell'=1}^{\ell} \mathbf{1}\{\pi_{\ell'}(i) < \pi_{\ell'}(j)\}$ . In the (full) rank aggregation problem,  $\pi_1, \dots, \pi_\ell$  are full rankings. A well-known 2-approximation for full rank aggregation outputs one of the input permutations at random: the expected cost for pair  $i, j$  is  $2w_{(i,j)}w_{(j,i)}$  which is not more than  $2 \min\{w_{(i,j)}, w_{(j,i)}\}$ . It follows that returning the best input permutation is also a 2-approximation algorithm.

In the partial rank aggregation problem, we will denote by  $i \equiv j$  if  $w_{(i,j)} = w_{(j,i)} = 0$ , i.e. if none of the input rankings prefer  $i$  to  $j$  or vice versa. Ailon [2] proposes the algorithm RepeatChoice for partial rank aggregation. Let  $\pi_1, \dots, \pi_\ell$  be the input rankings;  $\pi$  will be our final output ranking. We start by setting  $\pi(i) = 1$  for all  $i \in V$ . Then we repeatedly choose an input ranking  $\pi_k$  uniformly at random without replacement; we check each  $i, j \in V$  and if  $\pi(i) = \pi(j)$  but  $\pi_k(i) < \pi_k(j)$ , we modify  $\pi$  so that now  $\pi'(i) < \pi'(j)$ . We can do this by setting  $\pi'(h) = \pi(h)$  if  $\pi(h) \leq \pi(i)$  and  $\pi'(h) = \pi(h) + 1$  if  $h = j$  or  $\pi(h) > \pi(i)$ . At the end of the algorithm, the only pairs that have  $\pi(i) = \pi(j)$

are the pairs such that  $i \equiv j$ . We break these ties arbitrarily, to obtain a full ranking.

Note that for  $i \not\equiv j$  the probability that  $i$  is ranked before  $j$  is  $\frac{w_{(i,j)}}{w_{(i,j)}+w_{(j,i)}}$  which incurs a cost of  $w_{(j,i)}$ . Since  $i$  is either ranked before  $j$ , or  $j$  before  $i$ , the expected cost for pair  $i, j$  such that  $i \not\equiv j$  is  $\frac{2w_{(j,i)}w_{(i,j)}}{w_{(i,j)}+w_{(j,i)}}$ , which is clearly at most  $2 \min\{w_{(i,j)}, w_{(j,i)}\}$ . Ailon shows that this algorithm can be derandomized.

Ailon, Charikar and Newman show that the best of their algorithm's solution and the best input permutation is a  $\frac{11}{7}$ -approximation algorithm for (full) rank aggregation. We show that a similar guarantee can be given for our deterministic algorithm, and moreover that this guarantee also holds for partial rank aggregation, i.e. the best of our algorithm's solution, and the solution given by RepeatChoice gives a combinatorial  $\frac{8}{5}$ -approximation algorithm for partial rank aggregation.

**Theorem 26** *There exists a deterministic combinatorial  $\frac{8}{5}$ -approximation algorithm for partial rank aggregation which runs in  $O(n^3)$  time.*

**Proof:** Consider a meta-algorithm that with probability  $\frac{2}{5}$  will run the FAS-Pivot algorithm, and with probability  $\frac{3}{5}$  will output the solution returned by RepeatChoice. If the expected cost of this algorithm is within  $\frac{8}{5}$  of the optimal value, then so is either the FAS-Pivot solution or the (derandomized) RepeatChoice solution. Hence returning the better of the FAS-Pivot solution and the RepeatChoice solution is a  $\frac{8}{5}$ -approximation.

Another way of looking at the meta-algorithm is that we run FAS-Pivot but charge  $\frac{2}{5}$  times the cost of the solution it generates, plus  $\frac{3}{5}$  times the expected cost of the permutation returned by RepeatChoice. In other words, we can think of the meta-algorithm as running FAS-Pivot on a new input with weights  $\tilde{w}_{(i,j)} = \frac{2}{5}w_{(i,j)} + \frac{3}{5}2\frac{w_{(i,j)}w_{(j,i)}}{w_{(i,j)}+w_{(j,i)}}$  for every  $i \not\equiv j$ . If  $i \equiv j$ , then  $\tilde{w}_{(i,j)} = \tilde{w}_{(j,i)} = 0$ . We will show that Theorem 23 implies that FAS-Pivot gives a  $\frac{8}{5}$ -approximation algorithm for this new input.

Note that the budgets should not be changed, since we are still trying to bound against a lower bound for the real instance, so  $c_{ij} = \min\{w_{(i,j)}, w_{(j,i)}\}$ . Let  $G = (V, A)$  again be the majority tournament with respect to the original weight  $w$ . The first condition of Theorem 23 is met with  $\alpha = \frac{8}{5}$ , since if  $(i, j) \in A$ , then  $w_{(j,i)} = c_{ij}$ , so if  $i \neq j$  then

$$\tilde{w}_{(j,i)} = \frac{2}{5}w_{(j,i)} + \frac{3}{5}2\frac{w_{(i,j)}w_{(j,i)}}{w_{(i,j)} + w_{(j,i)}} = \frac{2}{5}c_{ij} + \frac{6}{5}\frac{w_{(i,j)}c_{ij}}{w_{(i,j)} + c_{ij}} \leq \frac{8}{5}c_{ij},$$

and of course if  $i \equiv j$  then  $\tilde{w}_{(j,i)} = c_{ij} = 0$ .

To show the second condition is met, we let  $a_{ij} = w_{(i,j)} + w_{(j,i)}$ , and first rewrite  $\tilde{w}_{(i,j)}$  for  $(i, j) \in A$  with  $i \neq j$ :

$$\tilde{w}_{(i,j)} = \frac{2}{5}w_{(i,j)} + \frac{3}{5}2\frac{w_{(i,j)}w_{(j,i)}}{w_{(i,j)} + w_{(j,i)}} = \frac{2}{5}w_{(i,j)} + \frac{6}{5}\frac{w_{(i,j)}(a_{ij} - w_{(i,j)})}{a_{ij}} = \frac{8}{5}w_{(i,j)} - \frac{6}{5}\frac{w_{(i,j)}^2}{a_{ij}}.$$

Let  $\{(i, j), (j, k), (k, i)\}$  be a directed triangle in  $G$ . We want to show that  $\tilde{w}_{(i,j)} + \tilde{w}_{(j,k)} + \tilde{w}_{(k,i)} \leq \alpha(c_{ij} + c_{jk} + c_{ki})$ . Note that if  $i \equiv j$ , then  $\tilde{w}_{(i,j)} = c_{ij} = 0$ , hence  $i, j$  contributes nothing to either side of the inequality. Hence we need to show that  $\sum_{(g,h) \in t; g \neq h} (\frac{8}{5}w_{gh} - \frac{6}{5}\frac{w_{gh}^2}{a_{gh}}) \leq \frac{8}{5} \sum_{(g,h) \in t; g \neq h} c_{gh}$ .

By the triangle inequality on the weights,  $w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq w_{(i,j)} + w_{(j,k)} + (w_{(k,j)} + w_{(j,i)}) = a_{ij} + a_{jk}$ . Similarly, we get that  $w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq a_{ij} + a_{ki}$  and  $w_{(i,j)} + w_{(j,k)} + w_{(k,i)} \leq a_{jk} + a_{ki}$ . Adding these inequalities, we get that  $\sum_{(g,h) \in t} w_{gh} \leq \frac{2}{3} \sum_{(g,h) \in t} a_{gh}$ .

By these observations and since  $g \neq h$  if and only if  $a_{gh} > 0$ , we can apply Claim 27 below, and conclude that  $\sum_{(g,h) \in t; g \neq h} (8w_{gh} - 6\frac{w_{gh}^2}{a_{gh}}) \leq \sum_{(g,h) \in t; g \neq h} 8(a_{gh} - w_{gh}) = 8 \sum_{(g,h) \in t; g \neq h} c_{gh}$ , as required.  $\square$

**Claim 27** For  $w = (w_1, w_2, w_3)$ , and  $a = (a_1, a_2, a_3)$  such that  $0 \leq w_i \leq a_i \leq 1$  for

$i = 1, 2, 3$ , and  $\sum_{i=1}^3 w_i \leq \frac{2}{3} \sum_{i=1}^3 a_i$ :

$$\sum_{i:a_i>0} (16w_i - 6\frac{w_i^2}{a_i} - 8a_i) \leq 0.$$

**Proof:** Let  $f(w, a) = \sum_{i:a_i>0} (16w_i - 6\frac{w_i^2}{a_i} - 8a_i)$ , and let  $P = \{(w, a) : 0 \leq w_i \leq a_i \leq 1, \text{ for } i = 1, 2, 3, \text{ and } \sum_{i=1}^3 w_i \leq \frac{2}{3} \sum_{i=1}^3 a_i\}$ . We want to show that  $\max\{f(w, a) | (w, a) \in P\} \leq 0$ .

Note that  $P$  is a closed and bounded set, and  $f$  is continuous on  $P$ , hence  $f$  attains a maximum on  $P$ . We make the following observations:

- (i) The maximum has  $\sum_{i=1}^3 w_i = \frac{2}{3} \sum_{i=1}^3 a_i$ : Take any solution with  $\sum_{i=1}^3 w_i < \frac{2}{3} \sum_{i=1}^3 a_i$ . Then there exists some  $w_j < a_j$ . Let  $\delta > 0$  such that  $\delta \leq a_j - w_j$ ,  $\delta \leq \frac{2}{3} \sum_{i=1}^3 a_i - \sum_{i=1}^3 w_i$ . Note that since  $0 < w_j < a_j$ ,  $2w_j < 2a_j < \frac{8}{3}a_j$ , therefore we can choose  $\gamma > 0$  with  $\gamma < \min(\delta, \frac{8}{3}a_j - 2w_j)$ . Increasing  $w_j$  by  $\gamma$  changes  $f$  by  $16\gamma - 6\frac{\gamma^2 + 2\gamma w_j}{a_j} = \gamma(16 - 6\frac{\gamma + 2w_j}{a_j}) > 0$ , where the final inequality follows since  $\gamma + 2w_j < \frac{8}{3}a_j$ .
- (ii) We now consider a fixed  $\bar{a}$ , with  $\bar{a}_i \geq 0$  for  $i = 1, 2, 3$ . Then maximizing  $f(w, \bar{a})$  subject to  $\sum_{i=1}^3 w_i = \frac{2}{3} \sum_{i=1}^3 \bar{a}_i$  reduces to minimizing  $\sum_{i:\bar{a}_i>0} \frac{w_i^2}{\bar{a}_i}$ . Since  $\sum_{i:\bar{a}_i>0} \frac{w_i^2}{\bar{a}_i}$  is a convex function of  $w$ , and  $\sum_{i=1}^3 w_i = \frac{2}{3} \sum_{i=1}^3 \bar{a}_i$  is linear, a necessary and sufficient condition for  $w$  to be a global minimum is (i)  $\sum_{i=1}^3 w_i = \frac{2}{3} \sum_{i=1}^3 \bar{a}_i$  and (ii) there exists some  $c$  such that  $\frac{2w_i}{\bar{a}_i} = c$  for every  $i$  such that  $\bar{a}_i > 0$ . This implies that the global minimum is achieved at  $w_i = \frac{2}{3}\bar{a}_i$  for every  $i$ .

It remains to note that  $f(w, a) = 0$  if  $w_i = \frac{2}{3}a_i$  for  $i = 1, 2, 3$ . □

### 3.2.3 Weighted Feedback Arc Set with Probability Constraints

In the previous subsection we used  $\sum_{i \in V, j \in V, i < j} \min\{w_{(i,j)}, w_{(j,i)}\}$  as the lower bound or budget in Theorem 23. If we only know that the weights satisfy the probability constraints, this lower bound has the problem that it is possible that  $\min\{w_{(i,j)}, w_{(j,i)}\} = 0$  for all  $i, j$ . We will therefore now turn to a linear programming relaxation, which can provide a better lower bound. If we let  $x_{(i,j)} = 1$  denote that  $i$  is ranked before  $j$ , then any feasible ranking satisfies  $x_{(i,j)} + x_{(j,i)} = 1$  and  $x_{(i,j)} + x_{(j,k)} + x_{(k,i)} \geq 1$  (since if  $x_{(i,j)} + x_{(j,k)} + x_{(k,i)} = 0$ , then  $j$  is ranked before  $i$ ,  $k$  is ranked before  $j$  but  $i$  is ranked before  $k$ , which is not possible). Hence the following linear program gives a lower bound on the minimum weight feedback arc set:

$$\begin{aligned}
 \min \quad & \sum_{i \in V, j \in V, i < j} (x_{(i,j)} w_{(j,i)} + x_{(j,i)} w_{(i,j)}) \\
 \text{s.t.} \quad & x_{(i,j)} + x_{(j,k)} + x_{(k,i)} \geq 1 && \text{for all distinct } i, j, k \\
 (LP_{FAS}) \quad & x_{(i,j)} + x_{(j,i)} = 1 && \text{for all } i \neq j \\
 & x_{(i,j)} \geq 0 && \text{for all } i \neq j.
 \end{aligned}$$

We use an optimal solution  $x$  to  $(LP_{FAS})$ , to give the budgets in Theorem 23, and in addition we use it to form the tournament we need in Theorem 23.

**Theorem 28** *There exists a deterministic 3-approximation algorithm for weighted minimum feedback arc set with probability constraints.*

**Proof:** Let  $x$  be an optimal solution to  $(LP_{FAS})$ . We let  $c_{ij} = x_{(i,j)} w_{(j,i)} + x_{(j,i)} w_{(i,j)}$ . We form a tournament  $G = (V, A)$ , where  $(i, j) \in A$  only if  $x_{(i,j)} \geq \frac{1}{2}$ . We show that the conditions in Theorem 23 hold with  $\alpha = 3$ . Since  $\sum_{i \in V, j \in V, i < j} c_{ij}$  is a lower bound on the weight of any feasible solution, this will imply the result.

The first condition in Theorem 23 holds with  $\alpha = 2$ , since for  $(i, j) \in A$ ,  $w_{(j,i)} \leq 2x_{(i,j)}w_{(j,i)} \leq 2c_{ij}$ .

Let  $t = \{(i, j), (j, k), (k, i)\}$  be a directed triangle in  $G$ . We need to show that  $\sum_{(g,h) \in t} w_{(g,h)} \leq 3 \sum_{(g,h) \in t} c_{gh}$ . We denote the left hand side by  $w(t)$  and the right hand side by  $3c(t)$ . We first rewrite the right hand side:

$$\begin{aligned} c(t) &= \sum_{(g,h) \in t} (w_{(h,g)}x_{(g,h)} + w_{(g,h)}(1 - x_{(g,h)})) \\ &= \sum_{(g,h) \in t} w_{(g,h)} + \sum_{(g,h) \in t} (w_{(h,g)} - w_{(g,h)})x_{(g,h)} \\ &= w(t) + \sum_{(g,h) \in t} (w_{(h,g)} - w_{(g,h)})x_{(g,h)}. \end{aligned}$$

Suppose without loss of generality that  $w_{(j,i)} - w_{(i,j)} = \min_{(g,h) \in t} \{w_{(h,g)} - w_{(g,h)}\}$ . To give a lower bound on  $c(t)$ , we consider the case that  $w_{(j,i)} - w_{(i,j)} \geq 0$  and the case that  $w_{(j,i)} - w_{(i,j)} < 0$ .

In the first case,  $w_{(h,g)} - w_{(g,h)} \geq 0$  for all  $(g, h) \in t$ . Hence  $c(t) = w(t) + \sum_{(g,h) \in t} (w_{(h,g)} - w_{(g,h)})x_{(g,h)} \geq w(t)$ . In the second case, we rewrite  $c(t)$  further as

$$c(t) = w(t) + \sum_{(g,h) \in t} \frac{1}{2} (w_{(h,g)} - w_{(g,h)}) + \sum_{(g,h) \in t} \left( x_{(g,h)} - \frac{1}{2} \right) (w_{(h,g)} - w_{(g,h)}).$$

By the definition of  $A$ ,  $x_{(g,h)} - \frac{1}{2} \geq 0$  for every  $(g, h) \in t$ , and we know from feasibility of  $x$  that  $\sum_{(g,h) \in t} x_{(g,h)} \leq 2$ . This means that if  $\min_{(g,h) \in t} \{w_{(h,g)} - w_{(g,h)}\} = w_{(j,i)} - w_{(i,j)} < 0$ , then  $c(t)$  obtains its lowest possible value if  $x_{(i,j)} = 1$  and  $x_{(j,k)} = x_{(k,i)} = \frac{1}{2}$ . Hence in this case

$$\begin{aligned} c(t) &\geq w(t) + (w_{(j,i)} - w_{(i,j)}) + \frac{1}{2}(w_{(k,j)} - w_{(j,k)}) + \frac{1}{2}(w_{(i,k)} - w_{(k,i)}) \\ &= w_{(j,i)} + \frac{1}{2}(w_{(k,j)} + w_{(j,k)}) + \frac{1}{2}(w_{(i,k)} + w_{(k,i)}). \end{aligned} \tag{3.3}$$

Since the weights satisfy the probability constraints,  $w_{(g,h)} + w_{(h,g)} = 1$ , and hence (3.3) is equal to  $1 + w_{(j,i)} \geq 1$ . By the probability constraints we also know that  $w(t) \leq 3$ , hence it follows that  $w(t) \leq 3c(t)$ .  $\square$

**Remark 6** *The LP based method from Theorem 28 can also be applied to give a 2-approximation algorithm for instances when the weights satisfy the triangle inequality.*

In the case when the weights satisfy the triangle inequality, the last part of the proof of Theorem 28 should be changed to: When the weights satisfy the triangle inequality,  $w_{(i,k)} + w_{(k,j)} \geq w_{(i,j)}$ , so the quantity in (3.3) is not less than  $w_{(j,i)} + \frac{1}{2}w(t) \geq \frac{1}{2}w(t)$ .

### 3.3 A Simple Clustering Algorithm

We will now show how to turn the algorithms from the previous sections into algorithms for correlation and consensus clustering. We use similar ideas as in Ailon et al. [4].

Instead of the majority tournament, they consider the complete undirected graph  $G = (V, E)$ , and partition the edge set into two sets  $E^+, E^-$  so that  $\{i, j\} \in E^+$  only if  $w_{\{i,j\}}^+ \geq w_{\{i,j\}}^-$ . As in [4] we will say this graph has a “bad triplet” if there exist vertices  $i, j, k$  such that  $\{i, j\} \in E^+, \{j, k\} \in E^+, \{k, i\} \in E^-$ . It is not hard to show that if the graph has no bad triplets, then there is a clustering that has  $i, j$  in the same cluster if  $\{i, j\} \in E^+$  and in separate clusters if  $\{i, j\} \in E^-$ . Clearly this is an optimal clustering, since for any vertex pair, the cost incurred in this clustering is  $\min\{w_{\{i,j\}}^+, w_{\{i,j\}}^-\}$ .

We give the algorithm CC-Pivot from [4] in Figure 3.2 (where in the version in [4] the pivot is chosen randomly from  $V$ ). We use the following notation: Given  $G = (V, E^+, E^-)$ , we denote by  $G(V')$  the subgraph of  $G$  induced by  $V' \subseteq V$ .

Ailon et al. [4] show that if the pivot is chosen uniformly at random from  $V$ , then CC-Pivot gives a 3-approximation if the weights satisfy probability constraints. It can also be shown using their analysis that CC-Pivot with a random pivot gives a 2-approximation if the weights satisfy the triangle inequality. As in the case of ranking, we show that we

**CC-Pivot**( $G = (V, E^+, E^-)$ )

Pick a pivot  $k \in V$ .

$C = \{k\} \cup \{i \in V : \{i, k\} \in E^+\},$

$R = \{i \in V : \{i, k\} \in E^-\}.$

Return  $\{C, \text{CC-Pivot}(G(R))\}.$

Figure 3.2: CC-Pivot Algorithm

can give a deterministic version of this algorithm with the same performance guarantees, by using a lower bound on the optimal value to guide our choice of pivot vertex.

Suppose a pair of vertices  $i, j$  is *not* ordered according to  $G$ . If  $j$  is clustered with  $i$  even though  $\{i, j\} \in E^-$ , then there is some recursive call in which both  $i$  and  $j$  end up in cluster  $C$ . If  $j$  is not clustered with  $i$  even though  $\{i, j\} \in E^+$ , then there is a recursive call in which  $i$  ends up in cluster  $C$ , and  $j$  is added to  $R$ . Neither  $i$  nor  $j$  can be the pivot in these cases. For a pivot  $k$ , let  $T_k^+(G)$  and  $T_k^-(G)$  be the sets of pairs for which these two things occur, if  $k$  is the pivot and the input to the recursive call is  $G$ , i.e.  $T_k^+(G) = \{\{i, j\} \in E^+ : \{j, k\} \in E^-, \{i, k\} \in E^+\}$  and  $T_k^-(G) = \{\{i, j\} \in E^- : \{j, k\} \in E^+, \{i, k\} \in E^+\}.$

We have a theorem similar to Theorem 23 in the previous section:

**Theorem 29** *Given an input  $(V, w^+, w^-)$  of the weighted clustering problem, a set of budgets  $\{c_{ij} : i \in V, j \in V, i \neq j\}$ , and a graph  $G = (V, E^+, E^-)$ , where  $E^+, E^-$  is a partition of  $\{\{i, j\} : i \in V, j \in V, i \neq j\}$  such that*

(i)  $w_{\{i,j\}}^- \leq \alpha c_{ij}$  for all  $\{i, j\} \in E^+$ , and

$w_{\{i,j\}}^+ \leq \alpha c_{ij}$  for all  $\{i, j\} \in E^-$ ,

(ii)  $w_{\{i,j\}}^+ + w_{\{j,k\}}^+ + w_{\{k,i\}}^- \leq \alpha(c_{ij} + c_{jk} + c_{ki})$  for every bad triplet  $\{i, j\} \in E^+, \{j, k\} \in$



$$E^+, \{k, i\} \in E^-.$$

Then CC-Pivot returns a solution that costs at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$  if we choose a pivot  $k$  that minimizes

$$\frac{\sum_{\{i,j\} \in T_k^+(G)} w_{\{i,j\}}^+ + \sum_{\{i,j\} \in T_k^-(G)} w_{\{i,j\}}^-}{\sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} c_{ij}}. \quad (3.4)$$

**Proof:** Let  $G = (V, E^+, E^-)$  be a graph that satisfies the conditions in the theorem. For a pair  $\{i, j\}$  with  $\{i, j\} \in E^+$ , we will let  $w_{ij} = w_{\{i,j\}}^+$  and  $\bar{w}_{ij} = w_{\{i,j\}}^-$ . For a pair  $\{i, j\}$  with  $\{i, j\} \in E^-$ , we let  $w_{ij} = w_{\{i,j\}}^-$  and  $\bar{w}_{ij} = w_{\{i,j\}}^+$ .

Then if a pair  $i, j$  is clustered according to  $G$ , the cost incurred is  $\bar{w}_{ij}$ , and for each pair not ordered according to  $G$ , the cost is  $w_{ij}$ . In order to stress the similarities with the ranking algorithm, we will call the first type of cost “forward cost” and the second “backward cost”.

The first condition implies that the forward cost is at most  $\alpha c_{ij}$ . As in the ranking case, we will show that the second condition implies that we always choose a pivot so that the backward cost incurred by pivoting on this vertex is at most  $\alpha$  times the budget for the vertex pairs involved.

For a given pivot  $k$ ,  $T_k^+(G) \cup T_k^-(G)$  is the set of pairs that incur a backward cost by pivoting on  $k$  when the set of vertices in the recursive call is  $V$ . Observe that if  $\{i, j\} \in T_k^+(G)$ , then  $i, j, k$  is a bad triplet, since  $\{i, j\} \in E^+$ ,  $\{i, k\} \in E^+$  and  $\{j, k\} \in E^-$ , and if  $\{i, j\} \in T_k^-(G)$ , then  $i, j, k$  is also a bad triplet, since  $\{i, j\} \in E^-$ ,  $\{i, k\} \in E^+$  and  $\{j, k\} \in E^+$ .

Therefore  $T_k^+(G) \cup T_k^-(G)$  contains exactly the pairs that are in a bad triplet with  $k$  in  $G$ . The cost incurred for the pairs in  $T_k^+(G) \cup T_k^-(G)$  if  $k$  is the pivot is equal to  $\sum_{\{i,j\} \in T_k^+(G)} w_{\{i,j\}}^+ + \sum_{\{i,j\} \in T_k^-(G)} w_{\{i,j\}}^- = \sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} w_{ij}$ , and we have a budget for these

vertex pairs of  $\sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} c_{ij}$ . The pivot chosen minimizes the ratio of the backward cost that is incurred to the budget for these pairs. We now show that the second condition implies that there exists a pivot for which this ratio is at most  $\alpha$ .

Let  $T$  be the set of bad triplets in  $G$ , and for a bad triplet with  $\{i, j\} \in E^+, \{j, k\} \in E^+, \{k, i\} \in E^-$ , let  $w(t) = w_{\{i,j\}}^+ + w_{\{j,k\}}^+ + w_{\{k,i\}}^- = w_{ij} + w_{jk} + w_{ki}$  and let  $c(t) = c_{ij} + c_{jk} + c_{ki}$ . If we sum  $\sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} w_{ij}$  over all  $k \in V$ , i.e.  $\sum_{k \in V} \sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} w_{ij}$ , then we count  $w_{ij}$  exactly once for every pivot  $k$  such that the vertex pairs  $\{i, j\}, \{j, k\}, \{k, i\}$  are in a bad triplet, hence  $\sum_{k \in V} \sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} w_{ij} = \sum_{t \in T} w(t)$ . Similarly,  $\sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} c_{ij} = \sum_{t \in T} c(t)$ .

By the second condition of the theorem,  $w(t) \leq \alpha c(t)$ . Therefore, there must exist some pivot  $k$  such that  $\sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} w_{ij} \leq \alpha \sum_{\{i,j\} \in T_k^+(G) \cup T_k^-(G)} c_{ij}$ , and the backward cost incurred when pivoting on  $k$  is not more than  $\alpha$  times the lower bound on the cost for the vertex pairs involved.  $\square$

**Remark 7** *The proof of Theorem 29 also implies that under the conditions of the theorem, choosing a pivot uniformly at random gives a solution with expected cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$ .*

**Lemma 30** *The algorithm in Theorem 29 can be implemented in  $O(n^3)$  time.*

**Proof:** The implementation can be done exactly as in the case of the ranking algorithm, except that we now maintain a list of the bad triplets rather than directed triangles for which all three vertices are in a single recursive call, and for each vertex we maintain the total backward cost incurred if pivoting on that vertex and the total “budget” for the vertex pairs involved, see Lemma 24.  $\square$

### 3.3.1 Weighted Clustering with Triangle Inequality

We now show how to use Theorem 29 to obtain a 2-approximation for clustering with triangle inequality. As in the case of rank aggregation, we then show that we can use the theorem plus the input clusterings to obtain a  $\frac{8}{5}$ -approximation algorithm for consensus clustering.

**Theorem 31** *There exists a deterministic combinatorial 2-approximation algorithm for weighted clustering with triangle inequality which runs in  $O(n^3)$  time.*

**Proof:** Partition the edges of the complete graph on  $V$  into  $E^+, E^-$  so that if  $\{i, j\} \in E^+$  then  $w_{\{i,j\}}^+ \geq w_{\{i,j\}}^-$  and if  $\{i, j\} \in E^-$  then  $w_{\{i,j\}}^- \geq w_{\{i,j\}}^+$ . Let  $c_{ij} = \min\{w_{\{i,j\}}^+, w_{\{i,j\}}^-\}$  for every  $i, j$ . Clearly  $\sum_{i \in V, j \in V, i < j} c_{ij}$  is a lower bound on the cost of any clustering of  $V$ .

Then the first condition of Theorem 29 is met with  $\alpha = 1$ .

Let  $\{i, j\} \in E^+, \{j, k\} \in E^+, \{k, i\} \in E^-$  be a bad triplet. By the triangle inequality on the weights,

$$w_{\{i,j\}}^+ \leq w_{\{j,k\}}^- + w_{\{k,i\}}^+ = c_{jk} + c_{ki}, \quad (3.5)$$

$$w_{\{j,k\}}^+ \leq w_{\{k,i\}}^- + w_{\{i,j\}}^+ = c_{ki} + c_{ij}, \quad (3.6)$$

$$w_{\{k,i\}}^- \leq w_{\{i,j\}}^- + w_{\{j,k\}}^- = c_{ij} + c_{jk}. \quad (3.7)$$

Adding these three inequalities implies the second condition of Theorem 29 holds for  $\alpha = 2$ . Hence Theorem 29 gives a deterministic combinatorial 2-approximation algorithm, which by Lemma 30 can be implemented in  $O(n^3)$  time.  $\square$

### 3.3.2 Consensus Clustering

In the case of consensus clustering, we can again do better than for the general problem of clustering with weights that satisfy the triangle inequality. Recall that we are given  $\ell$  collections  $C_1, \dots, C_\ell$ , where each  $C_k$  is a collection of disjoint subsets of  $V$ , such that  $w_{\{i,j\}}^+ = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbf{1}\{\exists C \in C_k \text{ s.t. } \{i, j\} \subset C\}$ , and  $w_{\{i,j\}}^- = \frac{1}{\ell} \sum_{k=1}^{\ell} \mathbf{1}\{\exists C \in C_k \text{ s.t. } |\{i, j\} \cap C| = 1\}$  where  $\mathbf{1}\{\cdot\}$  is the indicator function. We will first need a 2-approximation algorithm like the RepeatChoice algorithm for partial rank aggregation from [2].

Let  $C$  be our output clustering. We let  $S$  denote the vertices that are not contained in any set in  $C$ , i.e.  $S = V \setminus (\bigcup_{C \in C} C)$ . We start by setting  $C = \emptyset, S = V$ . We repeatedly choose an input clustering  $C_k$  uniformly at random without replacement; we consider the sets  $C \in C_k$  one by one, add the set  $C \cap S$  to the collection  $C$ , and update  $S$ . If at the end of this process,  $S$  is not empty, we add singleton sets to  $C$  for the remaining vertices in  $S$ . Note that if two vertices  $i, j$  are remaining in  $S$  after we have considered all input clusterings, then  $w_{\{i,j\}}^+ = w_{\{i,j\}}^- = 0$ , so we have no information about the similarity or dissimilarity of  $i, j$ . We will denote this by  $i \parallel j$ .

**Lemma 32** *Let  $C$  be the output of  $CC\text{-RepeatChoice}(V, C_1, \dots, C_\ell)$ . Then the expected cost of  $C$  is  $\sum_{\{i,j\}: i \not\parallel j} 2 \frac{w_{\{i,j\}}^+ w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$ .*

**Proof:** The probability that two vertices  $i, j, i \not\parallel j$  are in the same cluster of  $C$  is equal to the probability that among all input clusterings that contain either  $i$  or  $j$ , the first one chosen has both  $i$  and  $j$  in the same cluster, i.e. it's exactly  $\frac{w_{\{i,j\}}^+}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$ , and the probability that they are not in the same cluster of  $C$  is  $\frac{w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$ . It follows that the expected cost in  $C$  for pair  $i, j$  is  $2 \frac{w_{\{i,j\}}^+ w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$ .  $\square$

Since  $2 \frac{w_{\{i,j\}}^+ w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-} \leq 2 \min\{w_{\{i,j\}}^+, w_{\{i,j\}}^-\}$ ,  $CC\text{-RepeatChoice}$  is a 2-approximation algorithm.

This algorithm can be derandomized using standard techniques.

<p><b>CC-RepeatChoice</b>(<math>V, C_1, \dots, C_\ell</math>)</p> <hr/> <p>Initialize <math>C \leftarrow \emptyset, S \leftarrow V, K \leftarrow \{1, \dots, \ell\}</math>.</p> <p>While <math> K  &gt; 0</math></p> <div style="margin-left: 40px;"> <p>Choose <math>k \in K</math> uniformly at random.</p> <p><math>C \leftarrow C \cup \{C \cap S : C \in C_k\}</math>.</p> <p><math>S \leftarrow V \setminus \bigcup_{C \in C} C</math>.</p> <p><math>K \leftarrow K \setminus \{k\}</math>.</p> </div> <p>Return <math>C \cup \{\{i\} : i \in S\}</math>.</p>
---

Figure 3.3: CC-RepeatChoice Algorithm

**Theorem 33** *There exists a deterministic combinatorial  $\frac{8}{5}$ -approximation algorithm for partial consensus clustering that runs in  $O(n^3)$  time.*

**Proof:** As in the proof for Theorem 26, we analyze a virtual algorithm that with probability  $\frac{2}{5}$  runs CC-Pivot, and with probability  $\frac{3}{5}$  outputs the clustering found by CC-RepeatChoice. We will show that this algorithm gives a  $\frac{8}{5}$ -approximation algorithm, which implies that the best of the CC-Pivot and CC-RepeatChoice gives a  $\frac{8}{5}$ -approximation as well.

Since with probability  $\frac{3}{5}$ , the virtual algorithm outputs the clustering found by CC-RepeatChoice, we can think of the virtual algorithm as running CC-Pivot with weights  $\tilde{w}$ , where for  $i \nparallel j$ ,

$$\begin{aligned}\tilde{w}_{\{i,j\}}^+ &= \frac{2}{5}w_{\{i,j\}}^+ + \frac{3}{5}2\frac{w_{\{i,j\}}^+w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}, \\ \tilde{w}_{\{i,j\}}^- &= \frac{2}{5}w_{\{i,j\}}^- + \frac{3}{5}2\frac{w_{\{i,j\}}^+w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}.\end{aligned}$$

If  $i \parallel j$ , then we let  $\tilde{w}_{\{i,j\}}^+ = \tilde{w}_{\{i,j\}}^- = 0$ , since in that case both  $w_{\{i,j\}}^+ = 0$  and  $w_{\{i,j\}}^- = 0$ , so no cost is ever incurred for  $i, j$ .

We still have a lower bound of  $c_{ij} = \min\{w_{\{i,j\}}^+, w_{\{i,j\}}^-\}$  on the cost of clustering pair  $i, j$ , and we let  $G = (V, E^+, E^-)$  be the same as in the proof of Theorem 31, i.e.  $\{i, j\} \in E^+$  only if  $w_{\{i,j\}}^+ \geq w_{\{i,j\}}^-$  and  $\{i, j\} \in E^-$  only if  $w_{\{i,j\}}^- \geq w_{\{i,j\}}^+$  (breaking ties arbitrarily).

Let  $w_{ij} = \max\{w_{\{i,j\}}^+, w_{\{j,i\}}^-\}$ , and let  $a_{ij} = w_{\{i,j\}}^+ + w_{\{j,i\}}^- = w_{ij} + c_{ij}$ . We rewrite the weights in terms of this notation: If  $\{i, j\} \in E^+$ , then  $w_{\{i,j\}}^+ = w_{ij}$ ,  $w_{\{i,j\}}^- = c_{ij}$ , hence if  $i \nparallel j$

$$\begin{aligned}\tilde{w}_{\{i,j\}}^+ &= \frac{2}{5}w_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}, \\ \tilde{w}_{\{i,j\}}^- &= \frac{2}{5}c_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}.\end{aligned}$$

Similarly,  $\{i, j\} \in E^-$ , then  $w_{\{i,j\}}^+ = c_{ij}$ ,  $w_{\{i,j\}}^- = w_{ij}$ , and for  $i \nparallel j$ ,

$$\begin{aligned}\tilde{w}_{\{i,j\}}^+ &= \frac{2}{5}c_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}, \\ \tilde{w}_{\{i,j\}}^- &= \frac{2}{5}w_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}.\end{aligned}$$

We see that in this notation, if  $i \nparallel j$  then irrespective of whether the edge  $\{i, j\}$  is in  $E^+$  or  $E^-$ , the backward cost is  $\frac{2}{5}w_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}$  and the forward cost is  $\frac{2}{5}c_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}}$ .

The first condition of Theorem 29 is met with  $\alpha = \frac{8}{5}$ , since for  $i \nparallel j$ ,  $\frac{2}{5}c_{ij} + \frac{6}{5}\frac{w_{ij}c_{ij}}{w_{ij} + c_{ij}} \leq \frac{8}{5}c_{ij}$  and for  $i \parallel j$ ,  $\tilde{w}_{\{i,j\}}^+ = \tilde{w}_{\{i,j\}}^- = c_{ij} = 0$ .

To verify the second condition, let  $i, j, k$  be a bad triplet in  $G$  with  $\{i, j\} \in E^+$ ,  $\{j, k\} \in E^+$ ,  $\{k, i\} \in E^-$ , and let  $t = \{\{i, j\}, \{j, k\}, \{k, i\}\}$ . We need to show that

$$\tilde{w}_{\{i,j\}}^+ + \tilde{w}_{\{j,k\}}^+ + \tilde{w}_{\{k,i\}}^- \leq \frac{8}{5}(c_{ij} + c_{jk} + c_{ki}). \quad (3.8)$$

We rewrite the expression for the backward cost of  $g \nparallel h$  as

$$\frac{2}{5}w_{gh} + \frac{6}{5}\frac{w_{gh}c_{gh}}{w_{gh} + c_{gh}} = \frac{2}{5}w_{gh} + \frac{6}{5}\frac{w_{gh}(a_{gh} - w_{gh})}{a_{gh}} = \frac{8}{5}w_{gh} - \frac{6}{5}\frac{w_{gh}^2}{a_{gh}}.$$

Since pairs  $\{g, h\} \in t$  such that  $g \parallel h$  contribute 0 to both sides of (3.8), we need to show

$$\text{that } \sum_{\{g,h\} \in t: g \nparallel h} \left( \frac{8}{5}w_{gh} - \frac{6}{5}\frac{w_{gh}^2}{a_{gh}} \right) \leq \frac{8}{5} \sum_{(g,h) \in t: g \nparallel h} c_{gh}.$$

By adding inequalities (3.5), (3.6) and (3.7), we see that  $w_{ij} + w_{jk} + w_{ki} = w_{\{i,j\}}^+ + w_{\{j,k\}}^+ + w_{\{k,i\}}^- \leq 2(c_{ij} + c_{jk} + c_{ki})$ . Hence  $3(w_{ij} + w_{jk} + w_{ki}) \leq 2(c_{ij} + c_{jk} + c_{ki}) + 2(w_{ij} + w_{jk} + w_{ki}) = 2(a_{ij} + a_{jk} + a_{ki})$ . Hence the conditions of Claim 27 are satisfied, and therefore  $\sum_{\{g,h\} \in t: g \nparallel h} \left( \frac{8}{5} w_{gh} - \frac{6}{5} \frac{w_{gh}^2}{a_{gh}} \right) \leq \sum_{\{g,h\} \in t: g \nparallel h} \frac{8}{5} (a_{gh} - w_{gh})$  as required.  $\square$

### 3.3.3 Correlation Clustering

In the case of correlation clustering,  $\min\{w_{\{i,j\}}^+, w_{\{i,j\}}^-\}$  is not a strong enough lower bound on the cost incurred for pair  $\{i, j\}$  to use as  $c_{ij}$  in Theorem 29. Instead, we'll have to turn to a linear program to get a stronger lower bound. As in Section 3.2 for the case of weighted feedback arc set with probability constraints, we can use the optimal solution to a linear programming relaxation to provide the graph  $G = (V, E^+, E^-)$  and the budgets  $c_{ij}$  in Theorem 29.

Let  $x_{\{i,j\}}^+ = 1$  denote that  $i$  and  $j$  are in the same cluster,  $x_{\{i,j\}}^+ = 0$  that  $i$  and  $j$  are not in the same cluster, and let  $x_{\{i,j\}}^- = 1 - x_{\{i,j\}}^+$ . For three vertices  $i, j, k$ , it is impossible that  $i$  and  $j$  are in the same cluster ( $x_{\{i,j\}}^- = 0$ ),  $j$  and  $k$  are in the same cluster ( $x_{\{j,k\}}^- = 0$ ), but  $i$  and  $k$  are not in the same cluster ( $x_{\{i,k\}}^+ = 0$ ), hence for any feasible clustering  $x_{\{i,j\}}^- + x_{\{j,k\}}^- + x_{\{i,k\}}^+ \geq 1$ . The following linear program thus gives a lower bound on the value of an optimal clustering:

$$\begin{aligned}
& \min && \sum_{i \in V, j \in V, i < j} (x_{\{i,j\}}^+ w_{\{i,j\}}^- + x_{\{i,j\}}^- w_{\{i,j\}}^+) \\
& \text{s.t.} && x_{\{i,j\}}^- + x_{\{j,k\}}^- + x_{\{i,k\}}^+ \geq 1 && \text{for all distinct } i, j, k \\
& (LP_{CC}) && x_{\{i,j\}}^+ + x_{\{i,j\}}^- = 1 && \text{for all } i \neq j \\
& && x_{\{i,j\}}^+, x_{\{i,j\}}^- \geq 0 && \text{for all } i \neq j.
\end{aligned}$$

This linear program was also used by Ailon et al. [4], for the randomized rounding al-

gorithm which we will discuss in Section 3.5. We show that it can be used together with Theorem 29 to give a simple deterministic rounding that guarantees a 3-approximate solution.

**Theorem 34** *There exists a deterministic 3-approximation algorithm for correlation clustering.*

**Proof:** Let  $x$  be an optimal solution to  $(LP_{CC})$ . We will let  $c_{ij} = x_{\{i,j\}}^+ w_{\{i,j\}}^- + x_{\{i,j\}}^- w_{\{i,j\}}^+$ , and we partition the edges of the complete graph on  $V$  into  $E^+, E^-$  so that if  $\{i, j\} \in E^+$  then  $x_{\{i,j\}}^+ \geq x_{\{i,j\}}^-$ , and if  $\{i, j\} \in E^-$  then  $x_{\{i,j\}}^- \geq x_{\{i,j\}}^+$ .

The first condition in Theorem 29 holds with  $\alpha = 2$ , since if  $\{i, j\} \in E^+$ , then  $x_{\{i,j\}}^+ \geq \frac{1}{2}$ , hence  $c_{ij} = x_{\{i,j\}}^+ w_{\{i,j\}}^- + x_{\{i,j\}}^- w_{\{i,j\}}^+ \geq \frac{1}{2} w_{\{i,j\}}^-$ , and similarly if  $\{i, j\} \in E^-$ , then  $x_{\{i,j\}}^- \geq \frac{1}{2}$ , hence  $c_{ij} \geq \frac{1}{2} w_{\{i,j\}}^+$ .

To see that the second condition holds, let  $i, j, k$  be a bad triplet with  $\{i, j\} \in E^+, \{j, k\} \in E^+, \{k, i\} \in E^-$  and let  $t = \{\{i, j\}, \{j, k\}, \{k, i\}\}$ . Let  $w_{ij} = w_{\{i,j\}}^+, w_{jk} = w_{\{j,k\}}^+, w_{ki} = w_{\{k,i\}}^-$ , and let  $\bar{w}_{ij} = w_{\{i,j\}}^-, \bar{w}_{jk} = w_{\{j,k\}}^-, \bar{w}_{ki} = w_{\{k,i\}}^+$ ; in other words  $w_{gh}$  is the backward cost for pair  $\{g, h\}$  and  $\bar{w}_{gh}$  is the forward cost. We need to show that  $w_{ij} + w_{jk} + w_{ki} \leq 3(c_{ij} + c_{jk} + c_{ki})$ . Let  $x_{gh} = x_{\{g,h\}}^+$  if  $\{g, h\} \in E^+$ , and let  $x_{gh} = x_{\{g,h\}}^-$  otherwise. Then

$$c_{gh} = w_{gh}(1 - x_{gh}) + \bar{w}_{gh}x_{gh} = w_{gh} + (\bar{w}_{gh} - w_{gh})x_{gh}. \quad (3.9)$$

We also note that the feasibility of solution to  $(LP_{CC})$  implies that  $(1 - x_{ij}) + (1 - x_{jk}) + (1 - x_{ki}) \geq 1$ , or  $x_{ij} + x_{jk} + x_{ki} \leq 2$ .

Suppose without loss of generality that  $\bar{w}_{ij} - w_{ij} \leq \bar{w}_{jk} - w_{jk} \leq \bar{w}_{ki} - w_{ki}$ . To give a lower bound on  $c(t)$ , we consider the case that  $\bar{w}_{ij} - w_{ij} \geq 0$  and the case that  $\bar{w}_{ij} - w_{ij} < 0$ .



In the first case, also  $\bar{w}_{jk} - w_{jk} \geq 0$  and  $\bar{w}_{ki} - w_{ki} \geq 0$ , and hence we see from (3.9) that  $c_{ij} + c_{jk} + c_{ki} \geq w_{ij} + w_{jk} + w_{ki}$ .

On the other hand, if  $\bar{w}_{ij} - w_{ij} < 0$ , then we use the fact that  $x_{ij} + x_{jk} + x_{ki} \leq 2$ , and that each  $x$ -value is at least  $\frac{1}{2}$  by definition. If we look at

$$c_{ij} + c_{jk} + c_{ki} = w_{ij} + w_{jk} + w_{ki} + (\bar{w}_{ij} - w_{ij})x_{ij} + (\bar{w}_{jk} - w_{jk})x_{jk} + (\bar{w}_{ki} - w_{ki})x_{ki},$$

then we see that by our assumption that  $\bar{w}_{ij} - w_{ij}$  is the most negative value among the three, the smallest possible value this can take is if  $x_{ij} = 1, x_{jk} = \frac{1}{2}, x_{ki} = \frac{1}{2}$ , hence

$$\begin{aligned} c_{ij} + c_{jk} + c_{ki} &\geq w_{ij} + w_{jk} + w_{ki} + (\bar{w}_{ij} - w_{ij}) + \frac{1}{2}(\bar{w}_{jk} - w_{jk}) + \frac{1}{2}(\bar{w}_{ki} - w_{ki}) \\ &= \bar{w}_{ij} + \frac{1}{2}(w_{jk} + \bar{w}_{jk}) + \frac{1}{2}(w_{ki} + \bar{w}_{ki}). \end{aligned} \quad (3.10)$$

Since the weights satisfy the probability constraints,  $w_{gh} + \bar{w}_{gh} = 1$ , and hence the above is at least 1. But by the probability constraints we also know that  $w_{ij} + w_{jk} + w_{ki} \leq 3$ .  $\square$

**Remark 8** *The LP based method from Theorem 34 can also be applied to give a 2-approximation algorithm for instances when the weights satisfy the triangle inequality.*

In the case when the weights satisfy the triangle inequality, we note from equations (3.5), (3.6), (3.7) that  $\bar{w}_{jk} + \bar{w}_{ki} \geq w_{ij}$ , hence the quantity in (3.10) is not less than  $\frac{1}{2}(w_{ij} + w_{jk} + w_{ki})$ .

### 3.4 Constrained Problems

We now turn to constrained problems. In a constrained feedback arc set problem, we are given a partial order  $(V, P)$  in addition to  $(V, w)$ , and our output needs to be a linear

extension of  $P$ . We assume that the input is consistent with  $P$ , so that for  $(i, j) \in P$ ,  $w_{(j,i)} = 0$ .

In a constrained clustering problem we are given sets  $P^+, P^-$  such that any feasible clustering should have  $i, j$  in the same cluster if  $\{i, j\} \in P^+$  and in different clusters if  $\{i, j\} \in P^-$ . We assume that the input is consistent with  $P^+, P^-$ , so that for  $\{i, j\} \in P^+$ ,  $w_{\{i,j\}}^- = 0$  and for  $\{i, j\} \in P^-$ ,  $w_{\{i,j\}}^+ = 0$ . We also assume that  $P^+, P^-$  are consistent and transitive, i.e. that  $P^+ \cap P^- = \emptyset$  and if  $\{i, j\}, \{j, k\} \in P^+$  then  $\{k, i\} \in P^+$  and if  $\{i, j\} \in P^+, \{j, k\} \in P^-$  then  $\{k, i\} \in P^-$ .

### 3.4.1 Ranking and Clustering with Triangle Inequality

We begin by showing that in aggregation problems, and in general problems where the weights satisfy the triangle inequality, we can just do a “clean up” of any given solution to ensure that the constraints are satisfied, without increasing the cost of the solution. We thank Frans Schalekamp for suggesting that this might be the case.

**Lemma 35** *Given an input  $(V, w)$  of the weighted feedback arc set problem, with weights that satisfy the triangle inequality, a partial order  $P$  such that  $w_{(j,i)} = 0$  for  $(i, j) \in P$  and a permutation  $\pi$ , then we can find a permutation  $\pi'$  that is a linear extension of  $P$  and costs not more than  $\pi$ .*

**Proof:** Let  $(i, j) \in P$  and suppose  $\pi(j) < \pi(i)$ . We call such  $(i, j)$  violated. Let  $K(i, j)$  be the set of vertices  $k$  such that  $\pi(j) < \pi(k) < \pi(i)$ , and let  $(i^*, j^*)$  be a violated pair such that for any vertex  $k \in K(i^*, j^*)$  it is the case that  $(j^*, k) \notin P$  and  $(k, i^*) \notin P$ . Note that by transitivity of  $P$ , if a violated pair exists, then there exists a violated pair that satisfies this condition.

Consider the permutation  $\pi'$  we obtain by moving  $j^*$  to the position just after  $i^*$  with probability  $p = \frac{1}{2}$  or otherwise moving  $i^*$  to the position just before  $j^*$ . Note that  $(i^*, j^*)$  is not violated in  $\pi'$  and no new violations are created.

The expected difference in the cost of permutations  $\pi'$  and  $\pi$  is given by

$$\begin{aligned} & w_{(j^*, i^*)} - w_{(i^*, j^*)} + \frac{1}{2} \sum_{k \in K(i^*, j^*)} (w_{(j^*, k)} - w_{(k, j^*)} + w_{(k, i^*)} - w_{(i^*, k)}) \\ & \leq w_{(j^*, i^*)} - w_{(i^*, j^*)} + \frac{1}{2} \sum_{k \in K(i^*, j^*)} (2w_{(j^*, i^*)}) = -w_{(i^*, j^*)} \leq 0, \end{aligned}$$

where the inequality follows from the triangle inequality, since  $w_{(j^*, k)} \leq w_{(j^*, i^*)} + w_{(i^*, k)}$  and  $w_{(k, i^*)} \leq w_{(k, j^*)} + w_{(j^*, i^*)}$ , and the equality follows since  $w_{(j^*, i^*)} = 0$ . Hence either moving  $j^*$  to the position just after  $i^*$  or moving  $i^*$  to the position just before  $j^*$  does not increase the cost of the permutation, and has fewer violations.  $\square$

**Lemma 36** *Given an input  $(V, w^+, w^-)$  of the weighted clustering problem, with weights that satisfy the triangle inequality, constraints given by  $P^+, P^-$ , and a clustering  $C$  that does not satisfy  $P^+, P^-$ , then we can find a clustering  $C'$  that satisfies  $P^+, P^-$  and does not cost more than  $C$ .*

**Proof:** We will say a clustering  $C$  violates  $\{i, j\} \in P^+$  if it has  $i$  and  $j$  in different clusters, and we will say it violates  $\{i, j\} \in P^-$  if it has  $i$  and  $j$  in the same cluster. Consider a clustering  $C$  that violates  $\{i, j\} \in P^-$ . Let  $C$  be the cluster containing  $i$  and  $j$ . Let  $C_i = \{i\} \cup \{i' \in C : \{i, i'\} \in P^+\}$ , and similarly define  $C_j$ . Note that for  $i' \in C_i, j' \in C_j$ , it must be the case that  $w_{\{i', j'\}}^+ = 0$ , since  $\{i', j'\} \in P^-$  by transitivity.

Let  $R = C \setminus (C_i \cup C_j)$ . We randomly construct a clustering  $C'$  from  $C$  by making  $C_i$  and  $C_j$  separate clusters, and adding all vertices in  $R$  to  $C_i$  with probability  $\frac{|C_i|}{|C_i| + |C_j|}$  and to

$C_j$  with probability  $\frac{|C_j|}{|C_i|+|C_j|}$ . Then the expected difference in cost between  $C$  and  $C'$  is

$$\sum_{\substack{i' \in C_i \\ j' \in C_j}} (w_{\{i', j'\}}^+ - w_{\{i', j'\}}^-) + \frac{1}{|C_i| + |C_j|} \sum_{k \in R} \left( \sum_{j' \in C_j} |C_i| (w_{\{k, j'\}}^+ - w_{\{k, j'\}}^-) + \sum_{i' \in C_i} |C_j| (w_{\{k, i'\}}^+ - w_{\{k, i'\}}^-) \right).$$

The term  $\sum_{i' \in C_i, j' \in C_j} (w_{\{i', j'\}}^+ - w_{\{i', j'\}}^-)$  is nonpositive, since  $w_{\{i', j'\}}^+ = 0$  for all  $i' \in C_i, j' \in C_j$ .

We now fix some  $k \in R$  and consider

$$\frac{1}{|C_i| + |C_j|} \sum_{j' \in C_j} |C_i| (w_{\{k, j'\}}^+ - w_{\{k, j'\}}^-) + \sum_{i' \in C_i} |C_j| (w_{\{k, i'\}}^+ - w_{\{k, i'\}}^-).$$

We rewrite this as

$$\frac{1}{|C_i| + |C_j|} \sum_{i' \in C_i} \sum_{j' \in C_j} (w_{\{k, j'\}}^+ - w_{\{k, j'\}}^- + w_{\{k, i'\}}^+ - w_{\{k, i'\}}^-).$$

Now, note that  $w_{\{k, j'\}}^+ \leq w_{\{k, i'\}}^- + w_{\{i', j'\}}^+ = w_{\{k, i'\}}^-$  and similarly  $w_{\{k, i'\}}^+ \leq w_{\{k, j'\}}^-$ . Hence the second term is nonpositive also.

This implies that letting  $C'$  be the cheapest of the two clusterings does not increase the cost of the clustering. Moreover, no new violations are created, and at least one violation is removed. Repeatedly applying this procedure ensures that  $C'$  has no violations of  $P^-$ .

Now suppose  $C'$  still violates  $\{i, j\} \in P^+$ . Let  $C$  be the cluster containing  $i$  and  $D$  the cluster containing  $j$ . Let  $C_i = \{i\} \cup \{i' \in C : \{i, i'\} \in P^+\}$  and let  $C_j = \{j\} \cup \{j' \in D : \{j, j'\} \in P^+\}$ . By transitivity,  $\{i', j'\} \in P^+$  for every  $i' \in C_i, j' \in C_j$ .

We randomly construct a clustering  $C''$  obtained from  $C'$  by either adding  $C_i$  to  $D$  with probability  $\frac{|C_j|}{|C_i|+|C_j|}$  or adding  $C_j$  to  $C$  with probability  $\frac{|C_i|}{|C_i|+|C_j|}$ . Note that this does not create any new violations by transitivity of the constraints, and the assumption that no violations of  $P^-$  exist.

The expected difference in cost between  $C'$  and  $C''$  is

$$\sum_{i' \in C_i, j' \in C_j} (w_{\{i', j'\}}^- - w_{\{i', j'\}}^+) \quad (3.11)$$

$$+ \sum_{k \in C \setminus C_i} \frac{|C_j|}{|C_i| + |C_j|} \sum_{i' \in C_i} (w_{\{k, i'\}}^+ - w_{\{k, i'\}}^-) + \sum_{k \in C \setminus C_i} \frac{|C_i|}{|C_i| + |C_j|} \sum_{j' \in C_j} (w_{\{k, j'\}}^- - w_{\{k, j'\}}^+) \quad (3.12)$$

$$+ \sum_{k \in D \setminus C_j} \frac{|C_i|}{|C_i| + |C_j|} \sum_{j' \in C_j} (w_{\{k, j'\}}^+ - w_{\{k, j'\}}^-) + \sum_{k \in D \setminus C_j} \frac{|C_j|}{|C_i| + |C_j|} \sum_{i' \in C_i} (w_{\{k, i'\}}^- - w_{\{k, i'\}}^+). \quad (3.13)$$

The term in (3.11) is nonpositive, since  $\{i', j'\} \in P^+$ , so  $w_{\{i', j'\}}^- = 0$  for every  $i' \in C_i, j' \in C_j$ .

Now fix  $k \in C \setminus C_i$ , and note that

$$\begin{aligned} & \frac{|C_j|}{|C_i| + |C_j|} \sum_{i' \in C_i} (w_{\{k, i'\}}^+ - w_{\{k, i'\}}^-) + \frac{|C_i|}{|C_i| + |C_j|} \sum_{j' \in C_j} (w_{\{k, j'\}}^- - w_{\{k, j'\}}^+) \\ &= \frac{1}{|C_i| + |C_j|} \sum_{i' \in C_i} \sum_{j' \in C_j} (w_{\{k, i'\}}^+ - w_{\{k, i'\}}^- + w_{\{k, j'\}}^- - w_{\{k, j'\}}^+) \\ &\leq \frac{1}{|C_i| + |C_j|} \sum_{i' \in C_i} \sum_{j' \in C_j} (w_{\{k, j'\}}^- + w_{\{i', j'\}}^+ - w_{\{k, i'\}}^- + w_{\{k, i'\}}^- + w_{\{i', j'\}}^+ - w_{\{k, j'\}}^+) \\ &= 0. \end{aligned}$$

By symmetry, we thus find that (3.12) and (3.13) are both nonpositive. Hence replacing  $C'$  by the option with the smallest cost gives a clustering with fewer violations without increasing the cost. Repeating this procedure gives the result.  $\square$

### 3.4.2 Ranking and Clustering with Probability Constraints

For a constrained problem with general weights, we can give an extra set of conditions on the tournament  $G = (V, A)$  or the graph  $G = (V, E^+, E^-)$  in Theorems 23 and 29 that ensure that the solution returned by the FAS-Pivot and CC-Pivot algorithms satisfy a given set of constraints. Note that in this section we do not require that  $w_{(i, j)} = 0$  if  $(j, i) \in P$  or  $w_{\{i, j\}}^+ = 0$  if  $\{i, j\} \in P^-$ ,  $w_{\{i, j\}}^- = 0$  if  $\{i, j\} \in P^+$ .

**Lemma 37** *Given a partial order  $P$ , if the tournament  $G = (V, A)$  in Theorem 23 satisfies:*

1. *if  $(i, j) \in P$ , then  $(i, j) \in A$ ,*
2. *if  $(i, j) \in P$ , then there exists no  $k$  such that both  $(j, k) \in A, (k, i) \in A$ ,*

*then FAS-Pivot outputs a linear extension of  $P$ .*

**Proof:** It is clear from the proof of Theorem 23 that a pair  $i, j$  is not ordered according to the arc connecting them in the tournament  $G$  only if there is some directed triangle containing this arc. Hence the conditions of the lemma ensure that the solution is a linear extension of  $P$ . □

**Lemma 38** *Given sets  $P^+, P^- \subseteq \{\{i, j\} : i \in V, j \in V, i \neq j\}$ , if  $G = (V, E^+, E^-)$  in Theorem 23 satisfies:*

1.  *$\{i, j\} \in P^+$  implies that  $\{i, j\} \in E^+$  and  $\{i, j\} \in P^-$  implies that  $\{i, j\} \in E^-$ .*
2. *If  $\{i, j\} \in P^+$ , then there exists no  $k$  such that both  $\{j, k\} \in E^+, \{k, i\} \in E^-$ , and if  $\{i, j\} \in P^-$ , then there exists no  $k$  such that both  $\{j, k\} \in E^+, \{k, i\} \in E^+$ .*

*then CC-Pivot outputs a clustering that has  $i, j$  in the same cluster if  $\{i, j\} \in P^+$  and  $i, j$  in different clusters if  $\{i, j\} \in P^-$ .*

**Proof:** If  $\{i, j\} \in E^+$ , then the only case in which the clustering found by CC-Pivot will have  $i$  and  $j$  in different clusters is if there is some bad triplet containing  $i, j$  and the third vertex of the bad triplet is chosen as pivot. Similarly, if  $\{i, j\} \in E^-$  then  $i$  and  $j$  can only end up in the same cluster if there is a bad triplet containing  $i, j$ . Hence

the conditions of the lemma ensure that the clustering found by CC-Pivot satisfies the requirements given by  $P^+, P^-$ .  $\square$

Using these two lemmas, we can now show that the results in Theorem 28 and Theorem 34 for weighted problems with probability constraints also hold for constrained versions of these problems.

**Theorem 39** *The result in Theorem 28 also holds for constrained ranking problems.*

**Proof:** We only need to show that we can ensure that the tournament described in the proof of Theorem 28 satisfies the two conditions of Lemma 37.

Let  $P$  be the input partial order. We add the following constraints to  $(LP_{FAS})$ :

$$x_{(i,j)} = 1 \text{ for all } (i, j) \in P.$$

Let  $x$  be an optimal solution to  $(LP_{FAS})$ . Note that  $\sum_{i \in V, j \in V, i < j} (w_{(i,j)} x_{(j,i)} + w_{(j,i)} x_{(i,j)})$  still provides a lower bound on the optimal value. As in the proof of Theorem 28, we form a tournament  $G = (V, A)$  by including arc  $(i, j)$  only if  $x_{(i,j)} \geq \frac{1}{2}$ . Such a tournament satisfies the first condition of Lemma 37. We can ensure that we satisfy the second condition by being careful about breaking ties, as shown below.

Suppose  $(i, j) \in P$  and  $(i, j), (j, k), (k, i)$  is a directed triangle in  $G$ . Since  $(i, j) \in P$ ,  $x_{(i,j)} = 1$ , and if  $(j, k), (k, i)$  are in  $A$ , then we must have  $x_{(j,k)} \geq \frac{1}{2}, x_{(k,i)} \geq \frac{1}{2}$ . But by the first set of constraints of  $(LP_{FAS})$ ,  $x_{(i,k)} + x_{(k,j)} \geq 1 - x_{(j,i)} = 1$ , or  $1 - x_{(k,i)} + 1 - x_{(j,k)} \geq 1$ , hence  $x_{(k,i)}$  and  $x_{(j,k)}$  must be equal to  $\frac{1}{2}$ . Hence we can ensure that there are no directed triangles containing arcs in  $P$  by first labeling the vertices such that if  $(i, j) \in P$ , then  $label(i) < label(j)$ . In the case of a tie  $x_{(g,h)} = x_{(h,g)} = \frac{1}{2}$ , we add arc  $(g, h)$  to  $A$  if  $label(g) < label(h)$ . Now an arc  $(i, j) \in P$  cannot be in a directed triangle

$(i, j), (j, k), (k, i)$  in  $A$ , since if it were then  $label(j) < label(k)$  and  $label(k) < label(i)$ , which contradicts the property of the labeling that  $(i, j) \in P$  implies  $label(i) < label(j)$ .

□

**Theorem 40** *The result in Theorem 34 also holds for constrained clustering problems.*

**Proof:** We only need to show that we can ensure that the partition of the edges into  $E^+, E^-$  described in the proof of Theorem 34 satisfies the two conditions of Lemma 38. Let  $P^+, P^-$  be the sets of vertex pairs that need to be in the same cluster respectively in different clusters. We add the following set of constraints to  $(LP_{CC})$ :

$$x_{\{i,j\}}^+ = 1 \text{ for all } \{i, j\} \in P^+,$$

$$x_{\{i,j\}}^- = 1 \text{ for all } \{i, j\} \in P^-.$$

Note that we can do this while maintaining that the optimal value of  $(LP_{CC})$  gives a lower bound on the cost of the optimal clustering.

We label the vertices in  $V$  such that  $\{i, j\} \in P^+ \Rightarrow label(i) = label(j)$  and  $\{i, j\} \in P^- \Rightarrow label(i) \neq label(j)$ . Note that this can be done by having one label for each connected component of the graph  $(V, P^+)$ , and giving this label to all the vertices in the component. We now partition the edges into  $E^+, E^-$  as in the proof of Theorem 34 except that in the case when  $x_{\{i,j\}}^+ = x_{\{i,j\}}^-$ , we no longer break ties arbitrarily, but add  $\{i, j\}$  to  $E^+$  if  $i$  and  $j$  have the same label, and otherwise we add  $\{i, j\}$  to  $E^-$ . The edge sets  $E^+, E^-$  satisfy the first condition of Lemma 38. We now verify that they satisfy the second condition.

Suppose  $\{i, j\} \in P^+$  and  $i, j$  and  $k$  form a bad triplet, i.e.  $\{i, j\} \in E^+, \{j, k\} \in E^+, \{k, i\} \in E^-$ . Then  $x_{jk}^+ \geq \frac{1}{2}, x_{ki}^- \geq \frac{1}{2}$ , but by the triangle inequality constraints of  $(LP_{CC})$  and the fact that  $x_{ij}^+ = 1$  if  $\{i, j\} \in P^+$ , we have  $x_{jk}^- + x_{ki}^+ \geq 1$ , or  $1 - x_{jk}^+ + 1 - x_{ki}^- \geq 1$ , so  $x_{jk}^+ = \frac{1}{2}$  and



$x_{ki}^- = \frac{1}{2}$ . But then  $label(j) = label(k)$  and  $label(k) \neq label(i)$ , so  $label(j) \neq label(i)$  which contradicts the properties of our labeling, since  $\{i, j\} \in P^+$ . A similar contradiction can be derived if we assume there is a bad triplet containing  $\{i, j\} \in P^-$ .  $\square$

### 3.5 Better LP Based Algorithms

Rounding the LP solution deterministically in Sections 3.2.3 and 3.3.3 does not yield an approximation algorithm with a ratio better than 2, since even the forward cost can be twice as high as the LP value for a pair of vertices. Ailon et al. [4] propose randomized rounding algorithms for weighted feedback arc set problems. After solving the  $(LP_{FAS})$  from Section 3.2.3, the solution is rounded by repeatedly choosing a pivot  $k$  at random, and partitioning the vertices in sets  $V_L$  and  $V_R$  as before, but now a vertex  $i$  is put into  $V_L$  with probability  $x_{(i,k)}$  and into  $V_R$  with probability  $x_{(k,i)} = 1 - x_{(i,k)}$ . A similar approach is proposed for weighted clustering, where the optimal solution to  $(LP_{CC})$  from Section 3.3.3 is rounded by repeatedly choosing a pivot  $k$  at random and partitioning the vertices into  $R$  and  $C$ , by including  $i$  in  $C$  with probability  $x_{\{i,k\}}^+$  and in  $R$  with probability  $x_{\{i,k\}}^- = 1 - x_{\{i,k\}}^+$ . Ailon [2] generalizes this randomized rounding algorithm for weighted feedback arc set with triangle inequality by perturbing the LP solution by a function  $h(x)$  that satisfies  $h(x) + h(1 - x) = 1$ , and using the perturbed LP solution as probabilities.

We now show how to extend the ideas from the previous sections to derandomize the randomized rounding algorithms in Ailon et al. [4], and the perturbed version in Ailon [2]. In particular, this allows us to obtain a deterministic  $\frac{5}{2}$ -approximation algorithm for ranking and clustering with probability constraints, and a  $\frac{3}{2}$ -approximation algorithm for ranking and clustering with triangle inequality. Combined with the ideas from Theorem 26 and Theorem 33, this also allows us to obtain a deterministic  $\frac{4}{3}$ -approximation al-

<b>FASLP-Pivot(<math>V, p</math>)</b> <hr/> Pick a pivot $k \in V$ . Set $V_L = \emptyset, V_R = \emptyset$ . For all $i \in V, i \neq k$ , with probability $p_{(i,k)}$ : add $i$ to $V_L$ , else (with probability $p_{(k,i)}$ ): add $i$ to $V_R$ . Return FASLP-Pivot( $V_L, p$ ), $k$ , FASLP-Pivot( $V_R, p$ ).
---

Figure 3.4: FASLP-Pivot Algorithm

gorithm for full rank aggregation and full consensus clustering, which matches the best randomized algorithms in [4].

### 3.5.1 Weighted Feedback Arc Set

We will now give a formal description of the LP rounding algorithms for weighted feedback arc set in [4] and [2]. We give the algorithm in a generalized form in Figure 3.4, where  $p = \{p_{(i,j)} : i \in V, j \in V, i \neq j\}$  satisfying  $p_{(i,j)} + p_{(j,i)} = 1$  gives the probabilities that a vertex is ordered to the left or right of the pivot vertex. In the algorithm in [4] and [2], the pivot is chosen at random from the vertices in  $V$ , and  $p$  is determined by an optimal solution  $x$  to  $(LP_{FAS})$  from Section 3.2.3.

Suppose  $j$  is ordered before  $i$  in the output of FASLP-Pivot. Note that this means that there was some recursive call that contained both  $i$  and  $j$  in which either (i) one of them was the pivot, and  $j$  was ordered to the left of  $i$ , or (ii) some vertex  $k \neq i, j$  was the pivot, and  $j$  was added to  $V_L$  and  $i$  was added to  $V_R$ . We will say that the cost  $w_{(i,j)}$  of ordering  $j$  before  $i$  is a forward cost in case (i), and we will say it is a backward cost

in case (ii). Note the difference from our previous definition of forward and backward costs. We will say a pair  $i, j$  gets *decided* in a particular iteration of FASLP-Pivot if it either incurs a forward cost (i.e. one of  $i, j$  is the pivot) or a backward cost (i.e. one of them gets assigned to  $V_L$  and one to  $V_R$ ).

Let  $T_k(V)$  be the set of arcs  $(i, j)$  that become backward in a recursive call on  $V$  when  $k \neq i, j$  is the pivot, i.e.  $j$  is in  $V_L$  and  $i$  is in  $V_R$ . Note that  $T_k(V)$  is a random set, since  $V_L, V_R$  are random sets. In particular, the probability that a particular arc  $(i, j)$  is in  $T_k(V)$  is  $p_{(j,k)}p_{(k,i)}$ . For notational convenience, we define  $p_{(j,i)}^k$  as the probability that  $j$  is ordered to the left of  $i$  when  $k$  is the pivot vertex, i.e.  $p_{(j,i)}^k = p_{(j,k)}p_{(k,i)}$ . Then the expected backward cost in the iteration is  $\mathbb{E}[\sum_{(i,j) \in T_k(V)} w_{(i,j)}] = \sum_{i \in V \setminus \{k\}, j \in V \setminus \{k\}; i \neq j} p_{(j,i)}^k w_{(i,j)}$ .

The algorithm has two sources of randomness: the pivot is chosen randomly from the vertex set, and the vertices are ordered to the left or right of the pivot according to given probabilities. We will derandomize the algorithm in two steps. We will again have a notion of a budget  $c_{ij}$  for each vertex pair, and we choose a pivot  $k$  such that ratio of the expected cost for the arcs in  $T_k(V)$  and  $\mathbb{E}[\sum_{(i,j) \in T_k(V)} c_{ij}]$  is as small as possible. Then we use the method of conditional expectations [23] to assign the vertices in  $V \setminus \{k\}$  to  $V_L$  or  $V_R$ . We start by analyzing the algorithm that chooses a pivot deterministically, but randomly assigns the vertices in  $V \setminus \{k\}$  to  $V_L$  and  $V_R$  according to the probabilities  $p_{(i,k)}, p_{(k,i)}$ . The following theorem states conditions under which this gives a solution within a factor  $\alpha$  of a given budget.

**Theorem 41** *Given an input  $(V, w)$  of the weighted feedback arc set problem, a probability matrix  $p$ , and budgets  $\{c_{ij} : i \in V, j \in V, i \neq j\}$ , such that*

$$(i) \quad p_{(i,j)}w_{(j,i)} + p_{(j,i)}w_{(i,j)} \leq \alpha c_{ij} \text{ for all } i, j,$$

(ii) for every distinct triple  $\{i, j, k\}$  in  $V$ ,

$$(p_{(i,j)}^k w_{(j,i)} + p_{(j,i)}^k w_{(i,j)}) + (p_{(j,k)}^i w_{(k,j)} + p_{(k,j)}^i w_{(j,k)}) + (p_{(k,i)}^j w_{(i,k)} + p_{(i,k)}^j w_{(k,i)}) \leq \alpha \left( (p_{(i,j)}^k + p_{(j,i)}^k) c_{ij} + (p_{(j,k)}^i + p_{(k,j)}^i) c_{jk} + (p_{(k,i)}^j + p_{(i,k)}^j) c_{ki} \right),$$

then FAS-LPPivot gives a solution with expected cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$  if we choose a pivot  $k$  that minimizes

$$\frac{\mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \right]}{\mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \right]}. \quad (3.14)$$

**Proof:** Under the first condition in the theorem, the expected cost for a pair that gets decided and incurs a forward cost in an iteration of FASLP-Pivot is at most  $\alpha$  times the budget for this pair.

On the other hand, if we show that there always exists a pivot for which the ratio in (3.14) is at most  $\alpha$ , then the expected combined cost of the pairs that get decided and incur a backward cost is at most  $\alpha$  times the expected budget for the pairs that incur a backward cost. Note that

$$\mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \right] = \sum_{i \in V \setminus \{k\}, j \in V \setminus \{k\}: i \neq j} p_{(j,i)}^k w_{(i,j)}.$$

Now consider  $\sum_{i \in V \setminus \{k\}, j \in V \setminus \{k\}: i \neq j} p_{(j,i)}^k w_{(i,j)}$ : we add  $p_{(j,i)}^k w_{(i,j)} + p_{(i,j)}^k w_{(j,i)}$  once for every pair  $\{i, j\}$  and vertex  $k \notin \{i, j\}$ . We can attribute this amount to the triple  $\{i, j, k\}$ . Then we see that to every triple of distinct vertices  $i, j, k$  we attribute a cost of

$$p_{(j,i)}^k w_{(i,j)} + p_{(i,j)}^k w_{(j,i)} + p_{(i,k)}^j w_{(k,i)} + p_{(k,i)}^j w_{(i,k)} + p_{(k,j)}^i w_{(j,k)} + p_{(j,k)}^i w_{(k,j)}. \quad (3.15)$$

For a triple  $t = \{i, j, k\}$  of distinct vertices, we will denote the quantity in (3.15) by  $w(t)$ . So if we let  $T$  be the set of all distinct triples of vertices, then  $\sum_{k \in V} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \right] = \sum_{t \in T} w(t)$ .

Similarly, in

$$\sum_{k \in V} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \right] = \sum_{k \in V} \sum_{i \in V \setminus \{k\}, j \in V \setminus \{k\}; i \neq j} p_{(i,j)}^k c_{ij},$$

we add  $p_{(j,i)}^k c_{ij} + p_{(i,j)}^k c_{ij}$  for every pair  $\{i, j\}$  and vertex  $k \notin \{i, j\}$ . If we again attribute this amount to the triple  $\{i, j, k\}$ , and let  $c(t)$  be the total budget attributed to triple  $t = \{i, j, k\}$ , then we see that

$$c(t) = (p_{(j,i)}^k + p_{(i,j)}^k) c_{ij} + (p_{(i,k)}^j + p_{(k,i)}^j) c_{ki} + (p_{(k,j)}^i + p_{(j,k)}^i) c_{jk}. \quad (3.16)$$

and  $\sum_{k \in V} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \right] = \sum_{t \in T} c(t)$ .

Now, note that by the second condition of the theorem  $w(t) \leq \alpha c(t)$ , and we conclude that

$$\sum_{k \in V} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \right] \leq \alpha \sum_{k \in V} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \right],$$

and hence a vertex  $k$  for which the ratio in (3.14) is at most  $\alpha$  exists.  $\square$

**Remark 9** *The proof of Theorem 41 also implies that FASLP-Pivot with a randomly chosen pivot gives a solution with expected cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$ .*

We now know that we can choose a pivot deterministically, but to give a deterministic algorithm, we need to assign the vertices to  $V_L$  and  $V_R$  deterministically, instead of randomly as in FASLP-Pivot. We can achieve this by using the method of conditional expectations [23].

**Theorem 42** *Under the conditions in Theorem 41, there exists a deterministic algorithm for weighted feedback arc set that returns a solution of cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$ .*

**Proof:** We define the following notation: Let  $V_L, V_R, V'$  be a partition of  $V \setminus \{k\}$ , and let  $\mathbb{E}[W_k(V) | V_L, V_R]$  be the expected total cost incurred in an iteration of FASLP-Pivot for

the arcs that get decided in that iteration when pivoting on  $k$  conditioned on the vertices in  $V_L$  and  $V_R$  being ordered to the left and right of  $k$  respectively and the vertices in  $V'$  are ordered left or right with probability  $p_{(i,k)}$  and  $p_{(k,i)}$ . Let  $\mathbb{E}[C_k(V)|V_L, V_R]$  be the expected total budget  $c_{ij}$  for the vertex pairs  $i, j$  that get decided in an iteration of FASLP-Pivot when pivoting on  $k$ , again conditioned on  $V_L, V_R$ . Note that the conditional expected backward cost is

$$\begin{aligned} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \middle| V_L, V_R \right] = & \sum_{j \in V_L, i \in V_R} w_{(i,j)} + \sum_{j \in V', i \in V': i \neq j} p_{(j,i)}^k w_{(i,j)} \\ & + \sum_{j \in V', i \in V_R} p_{(j,k)} w_{(i,j)} + \sum_{j \in V_L, i \in V'} p_{(k,i)} w_{(i,j)}, \end{aligned}$$

and the conditional expected budget for these vertex pairs is

$$\begin{aligned} \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \middle| V_L, V_R \right] = & \sum_{j \in V_L, i \in V_R} c_{ij} + \sum_{j \in V', i \in V': i \neq j} p_{(j,i)}^k c_{ij} \\ & + \sum_{j \in V', i \in V_R} p_{(j,k)} c_{ij} + \sum_{j \in V_L, i \in V'} p_{(k,i)} c_{ij}. \end{aligned}$$

Hence we can easily compute these conditional expectations and we get

$$\begin{aligned} \mathbb{E} [W_k(V) | V_L, V_R] = & \sum_{i \in V_L} w_{(k,i)} + \sum_{i \in V_R} w_{(i,k)} + \sum_{i \in V'} (p_{(i,k)} w_{(k,i)} + p_{(k,i)} w_{(i,k)}) \\ & + \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \middle| V_L, V_R \right], \end{aligned}$$

and

$$\mathbb{E} [C_k(V) | V_L, V_R] = \sum_{i \in V \setminus \{k\}} c_{ik} + \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} c_{ij} \middle| V_L, V_R \right].$$

Initializing  $V_L = \emptyset, V_R = \emptyset, V' = V \setminus \{k\}$ , then we have  $\mathbb{E} [W_k(V) | V_L, V_R] \leq \alpha \mathbb{E} [C_k(V) | V_L, V_R]$ , since

$$\mathbb{E} [W_k(V) | V_L = \emptyset, V_R = \emptyset] = \sum_{i \in V \setminus \{k\}} (p_{(i,k)} w_{(k,i)} + p_{(k,i)} w_{(i,k)}) + \mathbb{E} \left[ \sum_{(i,j) \in T_k(V)} w_{(i,j)} \right]$$

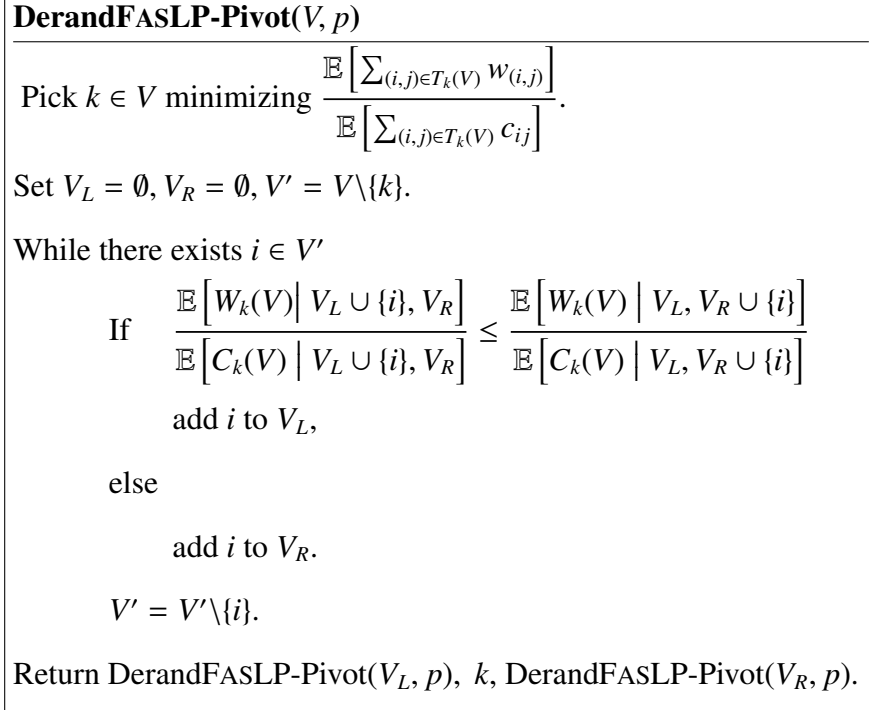


Figure 3.5: Derandomization of FASLP-Pivot.

and

$$\mathbb{E}\left[C_k(V) \mid V_L = \emptyset, V_R = \emptyset\right] = \sum_{i \in V \setminus \{k\}} c_{ik} + \mathbb{E}\left[\sum_{(i,j) \in T_k(V)} c_{ij}\right],$$

and by the first condition of Theorem 41  $p_{(i,k)}w_{(k,i)} + p_{(k,i)}w_{(i,k)} \leq \alpha c_{ik}$ , and by the second condition and the proof of Theorem 41,  $\mathbb{E}\left[\sum_{(i,j) \in T_k(V)} w_{(i,j)}\right] \leq \alpha \mathbb{E}\left[\sum_{(i,j) \in T_k(V)} c_{ij}\right]$ .

By standard conditional expectation arguments, we know that if we consider some vertex  $i \in V'$  and  $\mathbb{E}\left[W_k(V) \mid V_L, V_R\right] \leq \alpha \mathbb{E}\left[C_k(V) \mid V_L, V_R\right]$ , then we can add  $i$  to either  $V_L$  or  $V_R$  and maintain the invariant that  $\mathbb{E}\left[W_k(V) \mid V_L, V_R\right] \leq \alpha \mathbb{E}\left[C_k(V) \mid V_L, V_R\right]$ .

Hence we obtain the algorithm in Figure 3.5. By the invariant, at the end of the algorithm, we have a partition  $V_L, V_R$  of  $V \setminus \{k\}$  such that  $\mathbb{E}\left[W_k(V) \mid V_L, V_R\right] \leq \alpha \mathbb{E}\left[C_k(V) \mid V_L, V_R\right]$ . Since  $V_L, V_R$  partition  $V \setminus \{k\}$ ,  $\mathbb{E}\left[W_k(V) \mid V_L, V_R\right]$  is now equal to the (deterministic) cost of ordering the vertices in  $V_L$  and  $V_R$  to the left and right of  $k$

respectively, and  $\mathbb{E}[C_k(V) \mid V_L, V_R]$  is the total budget of the pairs that get decided in this iteration.  $\square$

**Corollary 43** *There exists a deterministic  $\frac{5}{2}$ -approximation algorithm for weighted feedback arc set with probability constraints.*

**Proof:** By Theorem 42, it suffices to show that we can give probabilities  $\{p_{(i,j)}\}$  and a lower bound  $\sum_{i \in V, j \in V, i < j} c_{ij}$  that satisfy the two conditions in Theorem 41 with  $\alpha = 2.5$ . Let  $x$  be an optimal solution to  $(LP_{FAS})$  in Section 3.2.3. Let  $p_{(i,j)} = x_{(i,j)}$ ,  $p_{(j,i)} = x_{(j,i)}$  and  $c_{ij} = x_{(i,j)}w_{(j,i)} + x_{(j,i)}w_{(i,j)}$  for every  $i, j \in V$ . Clearly, the first condition is satisfied with  $\alpha = 1$ . The second condition was shown to hold with  $\alpha = \frac{5}{2}$  in Lemma 13 in [4].  $\square$

**Corollary 44** *There exists a deterministic  $\frac{3}{2}$ -approximation algorithm for (constrained) weighted feedback arc set with triangle inequality.*

**Proof:** Note that by Lemma 35, any result we obtain for unconstrained problems, will also hold for constrained problems, so we may assume the problem is unconstrained. Let  $x$  be an optimal solution to  $(LP_{FAS})$  in section 3.2.3, and let

$$h(y) = \begin{cases} \frac{3}{4}y, & 0 \leq y \leq \frac{1}{3} \\ \frac{3}{2}y - \frac{1}{4}, & \frac{1}{3} < y \leq \frac{2}{3} \\ \frac{3}{4}y + \frac{1}{4}, & \frac{2}{3} < y \leq 1 \end{cases}$$

Let  $p_{(i,j)} = h(x_{(i,j)})$  and  $c_{ij} = x_{(i,j)}w_{(j,i)} + x_{(j,i)}w_{(i,j)}$  for every  $i, j \in V$ . These settings satisfy the conditions in Theorem 41 with  $\alpha = \frac{3}{2}$ , which was shown by Ailon in the proof of Theorem 4.1 in [2].  $\square$



**Corollary 45** *There exists a deterministic  $\frac{4}{3}$ -approximation algorithm for (constrained) full rank aggregation.*

**Proof:** We may assume by Lemma 35 that the problem is unconstrained.

We use the techniques from Theorem 26 to show that the best of DerandFASLP-Pivot and picking a random input permutation is within  $\frac{4}{3}$  of the optimal. If we output the result of DerandFASLP-Pivot with probability  $\frac{2}{3}$  and the best input permutation with probability  $\frac{1}{3}$ , then we can also think of the expected cost of the result as being the result when we run DerandFASLP-Pivot with  $\tilde{w}_{(i,j)} = \frac{2}{3}w_{(i,j)} + \frac{1}{3}(2w_{(i,j)}w_{(j,i)})$ . Let  $x$  be an optimal solution to the LP, and let  $c_{ij} = x_{(i,j)}w_{(j,i)} + x_{(j,i)}w_{(i,j)}$ , and let  $p_{(i,j)} = x_{(i,j)}$ .

Note that  $\tilde{w}_{(i,j)} \leq \frac{4}{3}w_{(i,j)}$  for any  $(i, j)$ , so

$$p_{(i,j)}\tilde{w}_{(j,i)} + p_{(j,i)}\tilde{w}_{(i,j)} \leq \frac{4}{3}(x_{(i,j)}w_{(j,i)} + x_{(j,i)}w_{(i,j)}) = \frac{4}{3}c_{ij},$$

and hence the first condition of Theorem 41 is satisfied for  $\alpha = \frac{4}{3}$ .

The second condition is exactly Lemma 14 in [4]. □

### 3.5.2 Weighted Clustering

We give a brief sketch of the randomized rounding approach, and subsequent derandomization, for the clustering case. Given  $p = \{p_{\{i,j\}}^+, p_{\{i,j\}}^- : i \in V, j \in V, i \neq j\}$  that satisfy  $p_{\{i,j\}}^+ + p_{\{i,j\}}^- = 1$ , the general form of the randomized rounding algorithm is given in Figure 3.6.

We will say a pair  $i, j$  incurs a forward cost if the vertices  $i, j$  were in the same recursive call in which one of them was the pivot, and we will say a pair  $i, j$  incurs a

<b>CCLP-Pivot(<math>V, p</math>)</b>
Pick a pivot $k \in V$ .
Set $C = \{k\}, V_R = \emptyset$ .
For all $i \in V, i \neq k$ ,
with probability $p_{\{i,k\}}^+$ : add $i$ to $C$ ,
else (with probability $p_{\{i,k\}}^-$ ): add $i$ to $R$ .
Return $\{C, \text{CCLP-Pivot}(R, p)\}$ .

Figure 3.6: CCLP-Pivot Algorithm

backward cost if the vertices  $i, j$  were in the same recursive call, in which some vertex  $k \neq i, j$  was the pivot, and  $i$  and  $j$  are not both in the next recursive call. We will say a pair  $i, j$  gets *decided* in a particular iteration of CCLP-Pivot if it either incurs a forward or a backward cost.

Note that there are two ways in which a pair can incur a backward cost: let  $T_k^+(V)$  be the pairs  $i, j$  such that exactly one of  $i, j$  is not in the next recursive call (for which we incur a cost  $w_{\{i,j\}}^+$ ), and let  $T_k^-(V)$  be the pairs  $i, j$  such that neither  $i$  nor  $j$  is in the next recursive call (incurring a cost  $w_{\{i,j\}}^-$ ). Let  $p_{\{i,j\},k}^- = p_{\{i,k\}}^- p_{\{j,k\}}^+ + p_{\{i,k\}}^+ p_{\{j,k\}}^-$ , and let  $p_{\{i,j\},k}^+ = p_{\{i,k\}}^+ p_{\{j,k\}}^+$ . Note that

$$\begin{aligned} \mathbb{E} \left[ \sum_{\{i,j\} \in T_k^+(V)} w_{\{i,j\}}^+ \right] &= \sum_{\{i,j\} \subset V \setminus \{k\}} p_{\{i,j\},k}^- w_{\{i,j\}}^+ \\ \mathbb{E} \left[ \sum_{\{i,j\} \in T_k^-(V)} w_{\{i,j\}}^- \right] &= \sum_{\{i,j\} \subset V \setminus \{k\}} p_{\{i,j\},k}^+ w_{\{i,j\}}^- \end{aligned}$$

We analyze the algorithm that chooses a pivot deterministically, but randomly assigns the vertices in  $V \setminus \{k\}$  to  $C$  and  $R$  according to the probabilities given by  $p$ . The following theorem states conditions under which this gives a solution within a factor  $\alpha$

of a given budget.

**Theorem 46** *Given an input  $(V, w^+, w^-)$  of the weighted clustering problem, probabilities  $p$ , budgets  $\{c_{ij} : i \in V, j \in V, i \neq j\}$  such that*

$$(i) \quad p_{\{i,j\}}^+ w_{\{i,j\}}^- + p_{\{i,j\}}^- w_{\{i,j\}}^+ \leq \alpha c_{ij} \text{ for all } i, j,$$

(ii) *for every distinct triple  $\{i, j, k\}$  in  $V$ ,*

$$\begin{aligned} & (p_{\{i,j\},k}^+ w_{\{i,j\}}^- + p_{\{i,j\},k}^- w_{\{i,j\}}^+) + (p_{\{j,k\},i}^+ w_{\{j,k\}}^- + p_{\{j,k\},i}^- w_{\{j,k\}}^+) + (p_{\{k,i\},j}^+ w_{\{k,i\}}^- + p_{\{k,i\},j}^- w_{\{k,i\}}^+) \leq \\ & \alpha \left( (p_{\{i,j\},k}^+ + p_{\{i,j\},k}^-) c_{ij} + (p_{\{j,k\},i}^+ + p_{\{j,k\},i}^-) c_{jk} + (p_{\{k,i\},j}^+ + p_{\{k,i\},j}^-) c_{ki} \right). \end{aligned}$$

*Then CC-Pivot returns a solution that costs at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$  if we choose a pivot  $k$  that minimizes*

$$\frac{\mathbb{E} \left[ \sum_{\{i,j\} \in T_k^+(V)} w_{\{i,j\}}^+ + \sum_{\{i,j\} \in T_k^-(V)} w_{\{i,j\}}^- \right]}{\mathbb{E} \left[ \sum_{\{i,j\} \in T_k^+(V) \cup T_k^-(V)} c_{ij} \right]}. \quad (3.17)$$

**Proof:** Under the first condition in the theorem, the expected cost for a pair that gets decided and incurs a forward cost in an iteration of FASLP-Pivot is at most  $\alpha$  times the budget for this pair.

As in the proof of Theorem 41 and Theorem 29, one can show that under the second condition, there always exists a pivot for which the ratio in (3.17) is at most  $\alpha$ . Hence the expected combined cost of the pairs that get decided and incur a backward cost is at most  $\alpha$  times the expected budget for the pairs that get decided and incur a backward cost.  $\square$

As in the ranking algorithm, given a pivot chosen according to (3.17), we can use the method of conditional expectations [23] to deterministically assign the vertices in  $V \setminus \{k\}$  to  $C$  and  $R$ , without increasing the ratio between the cost and the budget. The details are very similar to Theorem 42 and are omitted here.

**Theorem 47** *Under the conditions in Theorem 46, there exists a deterministic algorithm for weighted clustering that returns a solution of cost at most  $\alpha \sum_{i \in V, j \in V, i < j} c_{ij}$ .*

**Corollary 48** *There exists a deterministic  $\frac{5}{2}$ -approximation algorithm for correlation clustering.*

**Proof:** By Theorem 47, it suffices to show that we can give probabilities  $p$  and a lower bound  $\sum_{i \in V, j \in V, i < j} c_{ij}$  that satisfy the two conditions in Theorem 46 with  $\alpha = 2.5$ . Let  $x$  be an optimal solution to  $(LP_{CC})$  in section 3.3.3. Let  $p_{\{i,j\}}^+ = x_{\{i,j\}}^+$ ,  $p_{\{i,j\}}^- = x_{\{i,j\}}^-$  and  $c_{ij} = x_{\{i,j\}}^+ w_{\{i,j\}}^- + x_{\{i,j\}}^- w_{\{i,j\}}^+$  for every  $i, j \in V$ . Clearly, the first condition is satisfied with  $\alpha = 1$ . The second condition was shown to hold with  $\alpha = \frac{5}{2}$  in Lemma 15 in [4].  $\square$

We note that the result in Corollary 48 is close to the best possible result we can get if we use the optimal value of  $(LP_{CC})$  as a lower bound, since Charikar, Guruswami and Wirth [14] give the following example for which the integrality gap is 2: Given vertex set  $V = \{1, \dots, n\}$ , let  $w_{\{n,j\}}^+ = 1$  for  $j = 1, \dots, n-1$  and let  $w_{\{i,j\}}^+ = 0$  otherwise, and let  $w_{\{i,j\}}^- = 1 - w_{\{i,j\}}^+$ . Then the optimal fractional solution has  $x_{\{n,j\}}^+ = \frac{1}{2}$  for  $j = 1, \dots, n-1$  and  $x_{\{i,j\}}^+ = 0$  otherwise, with objective value  $\frac{1}{2}(n-1)$ . Any optimal clustering for this example places one of the vertices, say vertex 2, in a cluster with vertex 1, and all other vertices are in singleton clusters, which gives objective value  $(n-2)$ .

**Corollary 49** *There exists a deterministic  $\frac{4}{3}$ -approximation algorithm for (constrained) full consensus clustering.*

**Proof:** By Lemma 36, any approximation result for unconstrained consensus clustering implies the same for the constrained problem, so we can just consider unconstrained consensus clustering. Similar to Theorem 33, we charge DerandCCLP-Pivot  $\frac{2}{3}$  times

the cost of the clustering it generates, plus  $\frac{1}{3}$  times the expected cost of a randomly chosen input clustering. Let  $x$  be an optimal solution to  $(LP_{CC})$  in section 3.3.3. Let  $p_{\{i,j\}}^+ = x_{\{i,j\}}^+, p_{\{i,j\}}^- = x_{\{i,j\}}^-$  and  $c_{ij} = x_{\{i,j\}}^+ w_{\{i,j\}}^- + x_{\{i,j\}}^- w_{\{i,j\}}^+$  for every  $i, j \in V$ . We use Theorem 47 to show that the total charge to DerandCCLP-Pivot is at most  $\frac{4}{3} \sum_{i \in V, j \in V, i < j} c_{ij}$ . It then follows that either the cost of the clustering generated by DerandCCLP-Pivot, or the cost of the best input clustering, is at most  $\frac{4}{3} \sum_{i \in V, j \in V, i < j} c_{ij}$ .

To invoke Theorem 47, we need to show that the weights  $\tilde{w}$  defined as  $\tilde{w}_{\{i,j\}}^+ = \frac{2}{3} w_{\{i,j\}}^+ + \frac{2}{3} \frac{w_{\{i,j\}}^+ w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$  and  $\tilde{w}_{\{i,j\}}^- = \frac{2}{3} w_{\{i,j\}}^- + \frac{2}{3} \frac{w_{\{i,j\}}^+ w_{\{i,j\}}^-}{w_{\{i,j\}}^+ + w_{\{i,j\}}^-}$ , satisfy the conditions in Theorem 46. It is easily verified that the first condition of Theorem 46 is satisfied, and the second condition is exactly Lemma 16 in [4].  $\square$

### 3.6 Hierarchical Clustering

Recall that an  $M$ -level hierarchical clustering of a set  $V$  is a nested clustering of the elements in  $V$ , where the clustering at level  $m$  is a refinement of the clustering at level  $m + 1$ . Given a set  $V$  and a matrix  $D$  with  $D_{ij} \in \{0, \dots, M\}$  for any distinct  $i, j \in V$ , we want to find an  $M$ -level hierarchical clustering of  $V$  minimizing  $\sum_{i,j \in V} |D_{ij} - \lambda_{ij}|$ , where  $\lambda_{ij}$  is the number of levels in which  $i$  and  $j$  are in different clusters, or equivalently, since the clusterings are nested,  $i$  and  $j$  are in different clusters at levels  $1, \dots, \lambda_{ij}$ , and in the same cluster at levels  $\lambda_{ij} + 1, \dots, M$ . By Lemma 1(a) in [47] this is equivalent to finding an *ultrametric* that minimizes the  $\ell_1$  distance with  $D$ . An ultrametric is a tree metric in which all vertices are at the leaves of the tree, and the distance from each leaf to the root is the same.

A natural approach to the hierarchical clustering problem is to construct a hierarchical clustering in a top-down fashion: we first use a clustering algorithm to find the

top-level clustering  $C_M$ , and for each of the clusters in  $C_M$ , we call the clustering algorithm to find a refinement, which together form  $C_{M-1}$ , etcetera.

This approach is different from the approach used by Ailon and Charikar in [3]: although their algorithm is also a pivoting algorithm, it does not construct clusters top down, but instead uses the pivot to correct the distances of triples that violate the *ultrametric property*: For any ultrametric and for any distinct  $i, j, k$ , it is the case that  $D_{ij} \leq \max\{D_{jk}, D_{ik}\}$ . If  $k$  is the pivot, then the algorithm of Ailon and Charikar fixes the distances  $D_{ik}$  and  $D_{jk}$  to their current value, and adjusts  $D_{ij}$  to  $\max\{D_{ik}, D_{jk}\}$  if  $D_{ik} \neq D_{jk}$  or to  $\min\{D_{ik}, D_{jk}\}$  otherwise. The algorithm then recurses on the sets  $\{j \in V : D_{jk} = m\}$  for  $m = 1, \dots, M$ . The analysis is rather complicated and uses two different LP relaxations to give an upper bound on the cost of the algorithm's solution. Our algorithm and analysis are much simpler, and obtain the same approximation guarantee.

We start by redefining the input of the hierarchical clustering problem to make the connection to correlation clustering more obvious. For every pair of distinct vertices  $i, j$  and for every level  $m \in \{1, \dots, M\}$ , we let

$$w_{\{i,j\}}^{+,m} = \begin{cases} 1 & \text{if } D_{ij} < m \\ 0 & \text{otherwise,} \end{cases} \quad w_{\{i,j\}}^{-,m} = 1 - w_{\{i,j\}}^{+,m}. \quad (3.18)$$

Given an  $M$ -level hierarchical clustering and values  $\lambda_{ij}$  for every  $i, j$  such that  $i$  and  $j$  are in different clusters at levels  $1, \dots, \lambda_{ij}$ , and in the same cluster at levels  $\lambda_{ij} + 1, \dots, M$ , then

$$\begin{aligned} & \sum_{m=1}^M \left( w_{\{i,j\}}^{+,m} \mathbf{1}\{i, j \text{ separated at level } m\} + w_{\{i,j\}}^{-,m} \mathbf{1}\{i, j \text{ together at level } m\} \right) \\ &= \sum_{m=1}^M \left( w_{\{i,j\}}^{+,m} \mathbf{1}\{m \leq \lambda_{ij}\} + w_{\{i,j\}}^{-,m} \mathbf{1}\{m > \lambda_{ij}\} \right) \\ &= \sum_{m=1}^M \left( \mathbf{1}\{D_{ij} < m\} \mathbf{1}\{m \leq \lambda_{ij}\} + \mathbf{1}\{D_{ij} \geq m\} \mathbf{1}\{m > \lambda_{ij}\} \right) \\ &= |D_{ij} - \lambda_{ij}|. \end{aligned}$$

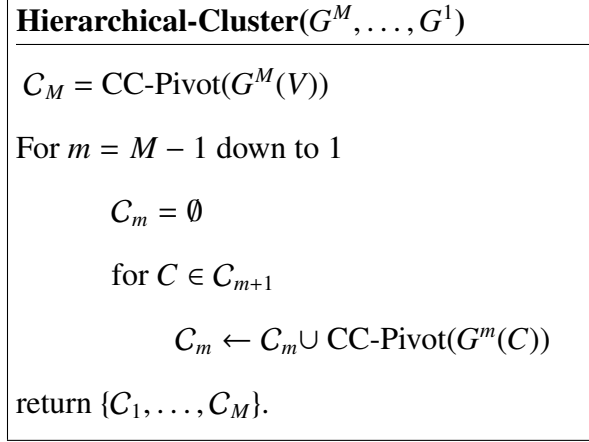


Figure 3.7: Algorithm for Hierarchical Clustering

Hence we can interpret the value  $w_{\{i,j\}}^{+,m}$  as the cost of having  $i$  and  $j$  in different clusters at level  $m$ , and  $w_{\{i,j\}}^{-,m}$  as the cost of having  $i$  and  $j$  in the same cluster at level  $m$ . At each level, we are solving a correlation clustering problem, with the additional requirement that the clusterings need to be nested.

We can now just think of calling the algorithm for correlation clustering from Section 3.3 for the top level, get a clustering  $C_M$  and then call CC-Pivot again for each cluster in  $C_M$  to get  $C_{M-1}$  etcetera. Hence we will need to have a partition of  $\{\{i, j\} : i \in V, j \in V, i \neq j\}$  into  $E^+$  and  $E^-$  for each of the  $M$  levels. Let these be given by  $G^m = (V, E^{+,m}, E^{-,m})$ .

We give our algorithm in Figure 3.7.

We cannot invoke Theorem 29 in this case, because if after the first call to CC-Pivot,  $i, j$  are in different clusters in  $C_M$ , then clearly  $i, j$  will also be in different clusters in  $C_{M-1}, \dots, C_1$ . Hence we will need to attribute a cost of  $\sum_{m=1}^M w_{\{i,j\}}^{+,m}$  if we separate  $i, j$  at the  $M$ -th level. Note that if  $i$  and  $j$  are in the same cluster in  $C_M$ , then we only need to attribute a cost of  $w_{\{i,j\}}^{-,M}$ .

We assume that we are given a budget  $c_{ij}^m$  for each pair of distinct vertices  $i, j$  and

each level  $m$ . Our next theorem will give conditions under which we can run HC-Pivot and find a hierarchical clustering that costs at most  $(M + 2) \sum_{m=1}^M \sum_{i \in V, j \in V, i < j} c_{ij}^m$ .

Let  $T_k^+(G)$  be defined as in Section 3.3. We will say that  $i, j, k$  are in a bad triplet on level  $m$  if  $\{i, j\} \in E^{+,m}, \{j, k\} \in E^{+,m}, \{k, i\} \in E^{-,m}$ . For notational convenience, we let  $W_{\{i,j\}}^{+,m} = \sum_{\ell=1}^m w_{\{i,j\}}^{+,\ell}$ , and  $C_{ij}^m = \sum_{\ell=1}^m c_{ij}^\ell$ .

**Theorem 50** *Given an input to the  $M$ -level hierarchical clustering problem, let  $(V, \{w_{\{i,j\}}^{+,m}, w_{\{i,j\}}^{-,m} : i \in V, j \in V, i \neq j\})$  for  $m = 1, \dots, M$  be inputs of weighted clustering derived according to (3.18). Given a set of budgets  $\{c_{ij}^m : i \in V, j \in V, i \neq j\}$ , and graphs  $G^m = (V, E^{+,m}, E^{-,m})$ , where  $E^{+,m}, E^{-,m}$  is a partition of  $\{\{i, j\} : i \in V, j \in V, i \neq j\}$  such that for every  $m \in \{1, \dots, M\}$ :*

- (i)  $w_{\{i,j\}}^{-,m} \leq \alpha c_{ij}^m$  for all  $\{i, j\} \in E^{+,m}$ , and  
 $W_{\{i,j\}}^{+,m} \leq \alpha C_{ij}^m$  for all  $\{i, j\} \in E^{-,m}$ ,
- (ii)  $W_{\{i,j\}}^{+,m} + W_{\{j,k\}}^{+,m} + w_{\{k,i\}}^{-,m} \leq \alpha(C_{ij}^m + C_{jk}^m + c_{ki}^m)$  for every bad triplet  $\{i, j\} \in E^{+,m}, \{j, k\} \in E^{+,m}, \{k, i\} \in E^{-,m}$ .

*Then Hierarchical-Cluster returns a solution that costs at most  $\alpha \sum_{m=1}^M \sum_{i \in V, j \in V, i < j} c_{ij}^m$  if in CC-Pivot( $G^m(C)$ ) we choose a pivot  $k$  that minimizes*

$$\frac{\sum_{\{i,j\} \in T_k^+(G^m(C))} W_{\{i,j\}}^{+,m} + \sum_{\{i,j\} \in T_k^-(G^m(C))} w_{\{i,j\}}^{-,m}}{\sum_{\{i,j\} \in T_k^+(G^m(C))} C_{ij}^m + \sum_{\{i,j\} \in T_k^-(G^m(C))} c_{ij}^m}. \quad (3.19)$$

**Proof:** In a call to CC-Pivot with graph  $G^m(C)$ , if we separate two vertices  $i, j$ , then they will be separated in the clusterings at levels  $1, \dots, m$ . Hence we attribute to this call a cost of  $\sum_{\ell=1}^m w_{\{i,j\}}^{+,\ell} = W_{\{i,j\}}^{+,m}$ . If two vertices  $i, j$  are in the same cluster, then we only attribute the cost  $w_{\{i,j\}}^{-,m}$  of joining them in the  $m$ -th level (since we do not yet know whether they will be in the same or in different clusters in the lower levels). We can also



attribute budgets to the call to CC-Pivot in the same way, attributing a budget of  $C_{ij}^m$  for a vertex pair  $i, j$  that gets separated, and a budget of  $c_{ij}^m$  for a vertex pair that gets clustered together.

We note that a piece of budget  $c_{ij}^m$  is attributed to exactly one call to CC-Pivot, and that the total cost of the solution generated by Hierarchical-Cluster is attributed.

It remains to show that the total cost attributed to a call to CC-Pivot is not more than  $\alpha$  times the total budget attributed to it. This follows immediately from Theorem 29.  $\square$

We will use the following linear program to demonstrate a set of budgets  $c^m$  and graphs  $G^m$  for  $m = 1, \dots, M$  that satisfy the conditions in Theorem 50. We let  $x_{\{i,j\}}^{+,m} = 1$  denote that  $i, j$  are in the same cluster at level  $m$ , and  $x_{\{i,j\}}^{-,m} = 1$  that they are in different clusters. We note that the first two sets of constraints are the same as in  $(LP_{CC})$ , and that the third set of constraints ensures that the clustering at level  $m - 1$  is a refinement of the clustering at level  $m$ .

$$\begin{aligned}
& \min \sum_{m=1}^M \sum_{i \in V, j \in V, i < j} \left( w_{\{i,j\}}^{+,m} x_{\{i,j\}}^{-,m} + w_{\{i,j\}}^{-,m} x_{\{i,j\}}^{+,m} \right) \\
& \text{s.t.} \quad x_{\{i,j\}}^{-,m} + x_{\{j,k\}}^{-,m} + x_{\{i,k\}}^{+,m} \geq 1 \quad \forall \text{ distinct } i, j, k, \forall m = 1, \dots, M \\
& (LP_{HC}) \quad x_{\{i,j\}}^{+,m} + x_{\{i,j\}}^{-,m} = 1 \quad \forall i \neq j, \forall m = 1, \dots, M \\
& \quad x_{\{i,j\}}^{-,m} \leq x_{\{i,j\}}^{-,m-1} \quad \forall i \neq j, \forall m = 2, \dots, M \\
& \quad x_{\{i,j\}}^{+,m} x_{\{i,j\}}^{-,m} \geq 0 \quad \forall i \neq j, \forall m = 1, \dots, M.
\end{aligned}$$

**Theorem 51** *There exists a deterministic  $(M + 2)$ -approximation algorithm for  $M$ -level hierarchical clustering.*

**Proof:** We form the graph  $G^m = (V, E^{+,m}, E^{-,m})$  exactly as we formed  $G = (V, E^+, E^-)$

in the proof of Theorem 34, by including  $\{i, j\} \in E^{+,m}$  if  $x_{\{i,j\}}^{+,m} \geq \frac{1}{2}$  and including  $\{i, j\} \in E^{-,m}$  if  $x_{\{i,j\}}^{-,m} \geq \frac{1}{2}$  (breaking ties arbitrarily).

We let  $c_{ij}^m = w_{\{i,j\}}^{+,m} x_{\{i,j\}}^{-,m} + w_{\{i,j\}}^{-,m} x_{\{i,j\}}^{+,m}$ . Then  $\sum_{m=1}^M \sum_{i \in V, j \in V, i < j} c_{ij}^m$  is a lower bound on the cost of the optimal hierarchical clustering. Hence if we can prove that the conditions in Theorem 50 hold with  $\alpha = M + 2$ , then our algorithm Hierarchical-Cluster, with pivots chosen as in Theorem 50, is an  $(M + 2)$ -approximation algorithm.

We claim that the first condition holds with  $\alpha = 2$ : If  $\{i, j\} \in E^{+,m}$ , then  $x_{\{i,j\}}^{+,m} \geq \frac{1}{2}$ , so  $c_{ij}^m \geq \frac{1}{2} w_{\{i,j\}}^{-,m}$ . If  $\{i, j\} \in E^{-,m}$ , then  $x_{\{i,j\}}^{-,m} \geq \frac{1}{2}$ , and by the third set of constraints this implies that also  $x_{\{i,j\}}^{-,\ell} \geq \frac{1}{2}$  for  $\ell = 1, \dots, m$ . Hence  $C_{ij}^m = \sum_{\ell=1}^m c_{ij}^\ell \geq \sum_{\ell=1}^m \frac{1}{2} w_{\{i,j\}}^{+,\ell} = \frac{1}{2} W_{\{i,j\}}^{+,m}$ .

We now show that the second condition holds with  $\alpha = m + 2$ , if considering a bad triplet in  $G^m$ . Since  $m \leq M$ , this implies that the second condition always holds with  $\alpha = M + 2$ . Let  $\{i, j\} \in E^{+,m}, \{j, k\} \in E^{+,m}, \{k, i\} \in E^{-,m}$  be a bad triplet. We note that we know from the proof of Theorem 29 that

$$w_{\{i,j\}}^{+,m} + w_{\{j,k\}}^{+,m} + w_{\{k,i\}}^{-,m} \leq 3(c_{ij}^m + c_{jk}^m + c_{ki}^m). \quad (3.20)$$

If  $m = 1$ , then this is exactly the second condition with  $\alpha = m + 2$ . Otherwise, let  $m \geq 2$ .

We will now show that for every  $\ell \in 1, \dots, m - 1$ :

$$w_{\{i,j\}}^{+,\ell} + w_{\{j,k\}}^{+,\ell} \leq (c_{ij}^m + c_{jk}^m + c_{ki}^m) + 4(c_{ij}^\ell + c_{jk}^\ell). \quad (3.21)$$

Adding (3.20) plus the inequalities (3.21) for  $\ell = 1, \dots, m - 1$ , then gives

$$\begin{aligned} W_{\{i,j\}}^{+,m} + W_{\{j,k\}}^{+,m} + W_{\{k,i\}}^{-,m} &= \sum_{\ell=1}^m w_{\{i,j\}}^{+,\ell} + \sum_{\ell=1}^m w_{\{j,k\}}^{+,\ell} + w_{\{k,i\}}^{-,m} \\ &\leq (m + 2)(c_{ij}^m + c_{jk}^m + c_{ki}^m) + 4 \left( \sum_{\ell=1}^{m-1} c_{ij}^\ell + \sum_{\ell=1}^{m-1} c_{jk}^\ell \right) \\ &\leq (m + 2) \left( \sum_{\ell=1}^m c_{ij}^\ell + \sum_{\ell=1}^m c_{jk}^\ell + c_{ki}^m \right) \\ &= (m + 2)(C_{ij}^m + C_{jk}^m + c_{ki}^m), \end{aligned}$$

where the last inequality follows since  $m \geq 2$ .

It remains to prove that inequality (3.21) holds. Note that the left hand side is either 0, 1 or 2. If it is 0, then clearly the inequality holds. If the left hand side is 1, suppose without loss of generality that  $w_{\{i,j\}}^{+, \ell} = 1$ ,  $w_{\{j,k\}}^{+, \ell} = 0$ . Note that from the definition  $w_{\{i,j\}}^{+, \ell} = 1$  means that  $D_{ij} < \ell$ . Then also  $D_{ij} < m$ , hence  $w_{\{i,j\}}^{+, m} = 1$ . In the proof of Theorem 34 we showed that either  $c_{ij}^m + c_{jk}^m + c_{ki}^m \geq w_{\{i,j\}}^{+, m} + w_{\{j,k\}}^{+, m} + w_{\{k,i\}}^{+, m}$ , or that  $c_{ij}^m + c_{jk}^m + c_{ki}^m \geq 1$ . Hence in this case  $c_{ij}^m + c_{jk}^m + c_{ki}^m$  is always at least 1, which proves that (3.21) holds if the left hand side is 1. Finally, we consider the case when the left hand side is 2. Then  $w_{\{i,j\}}^{+, \ell} = w_{\{j,k\}}^{+, \ell}$ , therefore

$$\begin{aligned}
c_{ij}^\ell + c_{jk}^\ell &= x_{\{i,j\}}^{-, \ell} + x_{\{j,k\}}^{-, \ell} \\
&\geq 1 - x_{\{k,i\}}^{+, \ell} && \text{(from the first set of constraints in } (LP_{HC})) \\
&= x_{\{k,i\}}^{-, \ell} && \text{(from the second set of constraints in } (LP_{HC})) \\
&\geq x_{\{k,i\}}^{-, m} && \text{(from the third set of constraints in } (LP_{HC})) \\
&\geq \frac{1}{2} && \text{(since } \{k, i\} \in E^{-, m}) \\
&= \frac{1}{4}(w_{\{i,j\}}^{+, \ell} + w_{\{j,k\}}^{+, \ell}).
\end{aligned}$$

□

We now turn to constrained hierarchical clustering: in addition to matrix  $D$ , we are given matrices  $L, U$  such that  $L_{ij} \leq D_{ij} \leq U_{ij}$  for every  $i, j \in V$ . Any feasible hierarchical clustering must have  $L_{ij} \leq \lambda_{ij} \leq U_{ij}$ , where  $\lambda_{ij}$  is again the number of levels in which  $i$  and  $j$  are in different clusters. We assume that the constraints are consistent, i.e. that for any  $i, j$ ,  $L_{ij} \leq U_{ij}$  and for any  $i, j, k$   $L_{ij} \leq \max\{U_{jk}, U_{ki}\}$ , since it is easy to check that otherwise no feasible hierarchical clustering exists.

**Lemma 52** *The result in Theorem 51 also holds for constrained hierarchical clustering.*

**Proof:** Let  $P^{+,m} = \{\{i, j\} : m > U_{ij}\}$ ,  $P^{-,m} = \{\{i, j\} : m \leq L_{ij}\}$  for  $m = 1, \dots, M$ . We add the following set of constraints to  $(LP_{CC})$

$$\begin{aligned} x_{\{i,j\}}^{+,m} &= 1 \text{ for all } \{i, j\} \in P^{+,m} \\ x_{\{i,j\}}^{-,m} &= 1 \text{ for all } \{i, j\} \in P^{-,m} \end{aligned}$$

Note that the optimal value of  $(LP_{HC})$  still gives a lower bound on the cost of the optimal hierarchical clustering. As in the proof of Theorem 40, we can then partition the edges into  $E^{+,m}, E^{-,m}$  so that the conditions of Lemma 38 are satisfied for  $m = 1, \dots, M$ . Hence by Lemma 38 our algorithm does not separate  $i, j$  at level  $m \in \{U_{ij} + 1, \dots, M\}$  since  $\{i, j\} \in P^{+,m}$  for  $m > U_{ij}$ . On the other hand, if  $i, j$  are not separated already at some level  $m > L_{ij}$ , then the fact that  $\{i, j\} \in P^{-,L_{ij}}$  ensures that  $i$  and  $j$  will be separated at level  $L_{ij}$  (and thus for all levels  $m \leq L_{ij}$ ). Hence the algorithm returns a feasible solution.  $\square$

Since CC-LPPivot gives a better approximation guarantee than CC-Pivot for correlation clustering, a natural question to ask is, whether we can get a better approximation guarantee if we use CC-LPPivot in Hierarchical-Cluster, instead of CC-Pivot. The following example shows that our analysis does not give a better approximation guarantee if we use CC-LPPivot.

Just as we adapted Theorem 29 for correlation clustering to Theorem 50 for hierarchical clustering, we could adapt Theorem 46 for use by the hierarchical clustering algorithm.

Given that  $i, j, k$  are in the same recursive call at level  $m$  and  $k$  is the pivot in this recursive call, let  $p_{\{i,j\},k}^{+,m} = x_{\{i,k\}}^{+,m} x_{\{j,k\}}^{+,m}$  be the probability  $i, j$  are clustered together at level  $m$ , and let  $p_{\{i,j\},k}^{-,m} = x_{\{i,k\}}^{-,m} x_{\{j,k\}}^{+,m} + x_{\{i,k\}}^{+,m} x_{\{j,k\}}^{-,m}$  be the probability that  $i, j$  are separated from level  $m$  (down to level 1).

The second inequality of Theorem 46 would then be that for every  $m \in 1, \dots, M$  and

for every distinct triple  $\{i, j, k\}$  in  $V$ ,

$$(p_{\{i,j\},k}^{+,m} w_{\{i,j\}}^{-,m} + p_{\{i,j\},k}^{-,m} W_{\{i,j\}}^{+,m}) + (p_{\{j,k\},i}^{+,m} w_{\{j,k\}}^{-,m} + p_{\{j,k\},i}^{-,m} W_{\{j,k\}}^{+,m}) + (p_{\{k,i\},j}^{+,m} w_{\{k,i\}}^{-,m} + p_{\{k,i\},j}^{-,m} W_{\{k,i\}}^{+,m}) \leq \\ \alpha \left( (p_{\{i,j\},k}^{+,m} + p_{\{i,j\},k}^{-,m}) c_{ij} + (p_{\{j,k\},i}^{+,m} + p_{\{j,k\},i}^{-,m}) c_{jk} + (p_{\{k,i\},j}^{+,m} + p_{\{k,i\},j}^{-,m}) c_{ki} \right).$$

Let  $D_{ij} = 0, D_{jk} = M, D_{ki} = M$ , so

$$w_{\{i,j\}}^{-,\ell} = 0, w_{\{i,j\}}^{+,\ell} = 1, \ell = 1, \dots, M,$$

$$w_{\{j,k\}}^{-,\ell} = 1, w_{\{j,k\}}^{+,\ell} = 0, \ell = 1, \dots, M,$$

$$w_{\{k,i\}}^{-,\ell} = 1, w_{\{k,i\}}^{+,\ell} = 0, \ell = 1, \dots, M,$$

and consider the following LP solution:

$$x_{\{i,j\}}^{-,\ell} = 0, x_{\{i,j\}}^{+,\ell} = 1, \ell = 1, \dots, M,$$

$$x_{\{j,k\}}^{-,M} = 1 - \epsilon, x_{\{j,k\}}^{+,M} = \epsilon, x_{\{j,k\}}^{-,\ell} = 1, x_{\{j,k\}}^{+,\ell} = 0, \ell = 1, \dots, M - 1,$$

$$x_{\{k,i\}}^{-,M} = 1 - \epsilon, x_{\{k,i\}}^{+,M} = \epsilon, x_{\{k,i\}}^{-,\ell} = 1, x_{\{k,i\}}^{+,\ell} = 0, \ell = 1, \dots, M - 1.$$

Then  $(p_{\{i,j\},k}^{+,M} w_{\{i,j\}}^{-,M} + p_{\{i,j\},k}^{-,M} W_{\{i,j\}}^{+,M}) + (p_{\{j,k\},i}^{+,M} w_{\{j,k\}}^{-,M} + p_{\{j,k\},i}^{-,M} W_{\{j,k\}}^{+,M}) + (p_{\{k,i\},j}^{+,M} w_{\{k,i\}}^{-,M} + p_{\{k,i\},j}^{-,M} W_{\{k,i\}}^{+,M}) = 2\epsilon(1 - \epsilon)M + 2\epsilon$  and  $(p_{\{i,j\},k}^{+,m} + p_{\{i,j\},k}^{-,m}) c_{ij} + (p_{\{j,k\},i}^{+,m} + p_{\{j,k\},i}^{-,m}) c_{jk} + (p_{\{k,i\},j}^{+,m} + p_{\{k,i\},j}^{-,m}) c_{ki} = 2\epsilon$ .

Hence we need  $\alpha \geq \frac{2\epsilon(1-\epsilon)M+2\epsilon}{2\epsilon} = (1 - \epsilon)M + 1$ .

## CHAPTER 4

# RANK AGGREGATION ON WEB DATA: POSITIONAL, COMPARISON SORT AND HYBRID ALGORITHMS

### 4.1 Introduction

We again consider the rank aggregation problem from Chapter 3, in which the goal is to find a ranking of a set of elements that best represents a given set of input rankings of the elements. In the previous chapter, we gave a deterministic combinatorial  $\frac{8}{5}$ -approximation algorithm for this problem, and an LP-based  $\frac{4}{3}$ -approximation algorithm. In this chapter, we consider the problem from a practical point of view. We test various known algorithms and propose new algorithms in order to achieve a good trade-off in computation time and performance.

There has been a lot of interest in this problem in the computer science community in recent years. The rank aggregation problem arises when building meta-search engines for Web search, where we want to combine the rankings obtained by different algorithms into a representative ranking. For example, Dwork, Kumar, Naor and Sivakumar [19] propose combining the rankings of individual search engines to get more robust rankings that are not sensitive to the various shortcomings and biases of individual search engines. Other applications arise in ranking movies, hotels, etc. based on the ratings given by users, or giving recommendations to a user based on several different criteria, where we can think of having one ranking of the alternatives for each criterion.

We repeat the formal definition of the rank aggregation problem from Chapter 3. Given two permutations  $\pi, \sigma$  of  $\{1, \dots, n\}$ , the Kendall tau distance is defined as

$$\mathcal{K}(\sigma, \pi) = \sum_{i=1}^n \sum_{j=i}^n \mathbf{1}\{(\pi(i) < \pi(j)) \& (\sigma(i) > \sigma(j))\}, \quad (4.1)$$

where  $\mathbf{1}\{\cdot\}$  is the indicator function. Given a set of permutations  $\pi_1, \dots, \pi_k$  of  $\{1, \dots, n\}$ , the Kemeny rank aggregation problem seeks a permutation  $\sigma$  that minimizes the number of pairwise disagreements with the input permutations, i.e.  $\frac{1}{k} \sum_{\ell=1}^k \mathcal{K}(\pi_\ell, \sigma)$ .

We also consider the *partial* rank aggregation problem. A partial ranking of  $V$  is a function  $\pi : V \rightarrow \{1, \dots, |V|\}$ , where the function  $\pi$  does not have to be one-to-one; in other words, a partial ranking is a ranking with ties. Fagin, Kumar, Ravi, Mahdian, Sivakumar, and Vee [25, 26, 27] proposed a set of different distance measures between two partial rankings and a permutation and a partial ranking. We will follow Ailon [2] and define the distance  $\mathcal{K}(\pi, \sigma)$  between two partial rankings  $\pi, \sigma$  as in Equation (4.1), and following Ailon we let the partial rank aggregation problem be the problem of finding a *permutation* that minimizes the sum of the distances to given input partial rankings. As in Chapter 3, we will use the term full rank aggregation if the input rankings are known to be permutations, and partial rank aggregation if the input rankings are partial rankings.

As we saw in Chapter 3, a problem related to rank aggregation is the feedback arc set problem. Given a tournament  $G = (V, A)$ , the feedback arc set problem asks for the smallest set of arcs  $A'$  such that  $(V, A \setminus A')$  is acyclic. Equivalently, we want to find a permutation of the vertices  $\pi$  that minimizes the number of “back-arcs”; the arcs that go from right to left if we order the vertices according to  $\pi$ . A generalization of this is the weighted feedback arc set problem, where given a set of vertices  $V$  and a nonnegative weight  $w_{(i,j)}$  for every ordered pair  $(i, j)$ , we want to find a permutation  $\pi$  that minimizes  $\sum_{i,j:\pi(i)<\pi(j)} w_{(j,i)}$ . We say the weights satisfy probability constraints if  $w_{(i,j)} + w_{(j,i)} = 1$  for all  $i, j \in V$ , and the triangle inequality if  $w_{(i,j)} + w_{(j,k)} \geq w_{(i,k)}$  for every  $i, j, k \in V$ .

If we let  $w_{(i,j)} = \frac{1}{k} \sum_{\ell=1}^k \mathbf{1}\{\pi_\ell(i) < \pi_\ell(j)\}$ , then rank aggregation is a special case of feedback arc set, since the weighted feedback arc set problem then seeks  $\sigma$  that

minimizes  $\sum_{i,j:\sigma(i)<\sigma(j)} \frac{1}{k} \sum_{\ell=1}^k \mathbf{1}\{\pi_\ell(i) < \pi_\ell(j)\} = \frac{1}{k} \sum_{\ell=1}^k \mathcal{K}(\pi_\ell, \sigma)$ . It is easily checked that the weights satisfy the triangle inequality. In the case of full rank aggregation, the weights also satisfy the probability constraints.

Several algorithms are known for rank aggregation with performance guarantees of 2 or less, see for example [4], [2], [17] and the previous chapter. For full rank aggregation, there is even a Polynomial-Time Approximation Scheme (PTAS) [53]: A PTAS is an algorithm that for any fixed  $\epsilon > 0$  finds a solution with performance guarantee  $(1 + \epsilon)$  in time polynomial in the size of the input, but not necessarily polynomial in  $\frac{1}{\epsilon}$ . In particular, the running time of the PTAS in [53] is doubly exponential in  $\frac{1}{\epsilon}$ .

Roughly speaking, many of the algorithms fall into one of three categories: positional methods, comparison sort methods, and hybrids of the two. A positional method for full rank aggregation seeks a permutation in which the *position* of each element is “close to the average position” of the element in the input permutations. Examples of positional methods are Borda’s method [11, 17], Footrule aggregation [18] and Pick-a-Perm [4]. Borda’s method finds a permutation that minimizes the distances between the elements’ positions and their mean positions in the input permutations, Footrule aggregation finds a permutation  $\sigma$  that minimizes  $\sum_{i \in V} \frac{1}{k} \sum_{\ell=1}^k |\pi_\ell(i) - \sigma(i)|$ , and Pick-a-Perm chooses one of the input permutations at random, thus returning a permutation in which the expected position of each element is its mean position. Comparison sort algorithms use a comparison relation to sort the elements, where the comparison relation is not necessarily transitive in this context. The result of a comparison sort algorithm hence depends on the comparison relation and the sorting algorithm. Note that the FAS-Pivot algorithm from Chapter 3 is an example of a QuickSort algorithm. Other examples of comparison sort methods are MergeSort and InsertionSort. To emphasize the fact that the FAS-Pivot algorithm from the previous chapter is a sorting algorithm, we will refer



to it as QuickSort in this chapter.

Both theoretically and practically there is evidence that hybrids of positional and comparison sort methods outperform the “pure” methods. For example, Ailon, Charikar and Newman [4] show that taking the best of the best input permutation, and the permutation obtained by running QuickSort, is a  $\frac{11}{7}$ -approximation algorithm. In the implementation study done by Dwork et al. [19], the best performing method was inspired by Copeland’s method [15] followed by InsertionSort, and Copeland’s method can itself be seen as a hybrid method.

In this chapter we study the three types of algorithms and propose new ways of combining positional and comparison sort algorithms. Positional algorithms have the advantage that they are fast, and have constant performance guarantees. However, we show that there exist examples where the solution returned by the positional algorithms has objective value at least twice the optimum. On the other hand, except for QuickSort, no performance guarantees are known for the comparison sort algorithms. And with the exception of the “deterministic QuickSort” algorithm of the previous chapter, which gives very good results but is quite slow, the comparison sort algorithms have the additional drawback of being randomized algorithms: the solution depends on random choices made by the algorithm. We therefore propose and evaluate new and existing hybrids of positional and comparison sort algorithms in the hope of finding the deterministic algorithms with a good trade-off in running time and performance.

The remainder of our chapter is organized as follows. In Section 4.1.1 we discuss some related papers that evaluate different algorithms for rank aggregation and related problems. In Section 4.2 we will discuss the positional and comparison sort algorithms that have been proposed in the literature, as well as existing and new hybrid methods. In Section 4.3 we show bad examples for the “pure” algorithms, and we show that the

hybrid methods perform much better on these examples. In Section 4.4 we evaluate the different algorithms on real data sets. We find that the hybrid methods give an excellent trade-off in running time and performance.

### 4.1.1 Related Work

There has been a lot of work in the algorithms community on the rank aggregation problem and the weighted feedback arc set problem with weights that satisfy the triangle inequality and/or probability constraints. Since we are comparing many of these algorithms, we defer the discussion of these works to the discussion of the algorithms we are considering. We mention some other studies that compare different algorithms for the rank aggregation problem or related problems.

Dwork et al. [19] propose aggregating the results from different search engines as a way to combat spam and get more robust search results. They investigate some traditional rank aggregation methods such as Footrule aggregation and Borda’s method, and they propose algorithms similar to PageRank, where they define a Markov chain on the search results and order the results based on their respective probabilities in the stationary distribution.

Gionis, Mannila, Puolamäki and Ukkonen [34] consider algorithms for a problem closely related to partial rank aggregation. They refer to a partial ranking as a bucket order, and the problem they consider is slightly different: they define the distance between two partial rankings  $\pi, \sigma$  as the number of pairs  $i, j$  such that  $\pi(i) < \pi(j)$  and  $\sigma(i) > \sigma(j)$  plus one half times the number of pairs that are tied in one of  $\pi, \sigma$  and not in the other. The goal is to find a partial ranking that minimizes the distance to given input partial rankings.

Coleman and Wirth [16] recently investigated the performance of different algorithms for the feedback arc set problem with weights that satisfy probability constraints. Although they consider rank aggregation as a special case, the algorithms they consider do not exploit the additional structure that rank aggregation problems have, and they do not consider any positional methods. They also show that some of the algorithms have very poor performance guarantees; however, the bad examples they give are not instances that could arise from a rank aggregation problem.

Finally, the problem of aggregating different clusters is closely related. In the consensus clustering problem, the input consists of  $k$  clusterings of a set of elements  $V$ , and the goal is to find a clustering of  $V$  that minimizes the number of pairwise disagreements with the input clusterings, i.e. the number of pairs that are in the same cluster in an input clustering but in different clusters in the output clustering or vice versa. A variation of the QuickSort algorithm also gives an approximation algorithm for consensus clustering [4]. Some papers that investigate the theoretical and practical performance of algorithms for consensus clustering are [35, 36].

## 4.2 Rank Aggregation Algorithms

We assume without loss of generality that the elements are numbered  $1, \dots, n$ . We think of a ranking  $\pi : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$  as a list of the elements  $\{1, \dots, n\}$  in order of preference (possibly with ties). We will thus say that a ranking  $\pi$  *prefers*  $i$  to  $j$ , or ranks  $i$  *higher* than  $j$ , if  $\pi(i) < \pi(j)$ . If  $\pi$  is a permutation, we will sometimes write it as  $\text{list}(\pi) = (\pi^{-1}(1), \pi^{-1}(2), \dots, \pi^{-1}(n))$ .

We assume that the input is provided in the form of  $k$  different permutations or partial rankings  $\pi_1, \dots, \pi_k$ , and multiplicities  $\mu_1, \dots, \mu_k$  (so that  $\pi_\ell$  occurs  $\mu_\ell$  times). We

let  $M = \sum_{\ell=1}^k \mu_{\ell}$ . Given such an input, we define an  $n$ -by- $n$  matrix  $w$ , where  $w_{(i,j)} = \sum_{\ell=1}^k \frac{\mu_{\ell}}{M} \mathbf{1}\{\pi_{\ell}(i) < \pi_{\ell}(j)\}$ .

### 4.2.1 Positional Algorithms

We call an algorithm for full rank aggregation *positional* if it seeks a permutation in which the *position* of each element is “close to the average position” of the element in the input permutations.

#### Borda

The Borda count of an element is its mean position in the input permutations, i.e.  $\text{Borda}(i) = \sum_{\ell=1}^k \frac{\mu_{\ell}}{M} \pi_{\ell}(i)$ . The Borda algorithm ranks the elements in order of increasing Borda counts. It was shown by [17] that this is equivalent to finding a permutation  $\sigma$  that minimizes  $\sum_{i \in V} |\sum_{\ell=1}^k \frac{\mu_{\ell}}{M} \pi_{\ell}(i) - \sigma(i)|$ , i.e. the sum of the distances from the elements’ position to their mean positions.

Note that in the corresponding instance to the weighted feedback arc set problem, ordering by Borda count is equivalent to ranking the vertices by increasing (weighted) indegree. For partial rank aggregation, we therefore define the Borda count of an element to be its weighted indegree in the corresponding tournament. It was shown by Coppersmith, Fleischer and Rudra [17] that this is a 5-approximation algorithm for the weighted feedback arc set problem if the weights satisfy the probability constraints; hence this holds for full rank aggregation.

We also consider RandBordaExp, a randomized Borda algorithm. Initially  $S = \{1, \dots, n\}$ , and  $p = 1$ . While  $p \leq n$ , we choose an element from  $S$  at random, where

element  $i$  is chosen with probability  $\exp(\text{Borda}(i)) / \sum_{j \in S} \exp(\text{Borda}(j))$ . We set  $\pi(p) = i$ , remove  $i$  from  $S$  and increase  $p$  by 1.

Of course, one could use different monotone functions of  $\text{Borda}(i)$  for the probability of choosing  $i$ , but we found that the above function worked well on our data sets.

## Footrule

The Spearman's Footrule distance between two permutations  $\pi, \sigma$  is defined as

$$\mathcal{F}(\pi, \sigma) = \sum_{i=1}^n |\pi(i) - \sigma(i)|.$$

It was shown by Diaconis and Graham [18] that

$$\mathcal{K}(\pi, \sigma) \leq \mathcal{F}(\pi, \sigma) \leq 2\mathcal{K}(\pi, \sigma).$$

Hence a permutation  $\sigma$  that minimizes the average Footrule distance  $\frac{1}{k} \sum_{\ell=1}^k \mathcal{F}(\sigma, \pi_\ell)$  to a given set of permutations  $\pi_1, \pi_2, \dots, \pi_k$  is a 2-approximation for the Kemeny rank aggregation problem. Finding a permutation  $\sigma$  that minimizes  $\frac{1}{k} \sum_{\ell=1}^k \mathcal{F}(\sigma, \pi_\ell)$  can be done in polynomial time by solving a bipartite matching problem (see Dwork et al. [19]). Dwork et al. also showed that if the median positions of the elements form a permutation, then this permutation is an optimal Footrule aggregation.

To extend Footrule aggregation to partial rank aggregation, we need to define the Footrule distance between a partial ranking and a full ranking. Given a partial ranking  $\pi$ , and an element  $i$  let  $\pi_{\min}(i) = \#\{j : \pi(j) < \pi(i)\}$  and let  $\pi_{\max}(i) = \#\{j : \pi(j) > \pi(i)\}$ , in other words,  $\pi_{\min}(i)$  and  $\pi_{\max}(i)$  are the minimum and maximum position that  $i$  could take in a full ranking  $\sigma$  such that  $\mathcal{K}(\pi, \sigma) = 0$ . We then define the Footrule distance between permutation  $\sigma$  and partial ranking  $\pi$  as

$$\mathcal{F}(\pi, \sigma) = \sum_{i=1}^n \min_{p \in [\pi_{\min}(i), \pi_{\max}(i)]} |p - \sigma(i)|.$$

We note that our definition of Footrule distance between a permutation and a partial ranking is different from the one proposed in Dwork et al. [19], since they use a different definition of a partial ranking. As in the case for full rank aggregation, we can find an optimal Footrule aggregation for the partial rank aggregation problem by solving a bipartite matching problem.

**Lemma 53** *Footrule aggregation is a 2-approximation algorithm for partial rank aggregation.*

**Proof:** We will show that for a permutation  $\sigma$  and a partial ranking  $\pi$ ,  $\mathcal{F}(\pi, \sigma) \leq 2\mathcal{K}(\pi, \sigma)$ .

Since we know that for a permutation  $\pi'$ ,  $\mathcal{F}(\pi', \sigma) \leq 2\mathcal{K}(\pi', \sigma)$ , it suffices to show that there exists a permutation  $\pi'$  such that  $\mathcal{K}(\pi', \sigma) = \mathcal{K}(\pi, \sigma)$  and  $\mathcal{F}(\pi', \sigma) \geq \mathcal{F}(\pi, \sigma)$ .

We let  $\pi'$  be the unique permutation that has  $\pi'(i) < \pi'(j)$  if  $\pi(i) < \pi(j)$  or if  $\pi(i) = \pi(j)$  and  $\sigma(i) < \sigma(j)$ . Clearly,  $\mathcal{K}(\pi', \sigma) = \mathcal{K}(\pi, \sigma)$ . On the other hand,  $\pi'(i) \in [\pi_{\min}(i), \pi_{\max}(i)]$  for any  $i$ , and hence  $\mathcal{F}(\pi', \sigma) \geq \mathcal{F}(\pi, \sigma)$ .  $\square$

### Pick-a-Perm

The Pick-a-Perm algorithm for full rank aggregation returns an input permutation at random. As we saw in Chapter 3, it is easily shown that this is a 2-approximation algorithm. This algorithm has a deterministic variant, where the input permutation that gives the smallest objective value is chosen. We call this the Best-of- $k$  algorithm.

Ailon [2] showed how to generalize the Pick-a-Perm algorithm to an algorithm for partial rank aggregation (we referred to this algorithm as RepeatChoice in Chapter 3):

We think of a partial ranking as a bucket order, where elements with the same rank are in one bucket. To construct output ranking  $\sigma$ , we start with  $\sigma$  having all elements in the same bucket. We repeatedly choose a partial ranking at random from the input rankings, and order the elements within each bucket of  $\sigma$  according to this partial ranking, until each bucket of  $\sigma$  is of size 1. Ailon shows that this algorithm is also a 2-approximation algorithm and can be derandomized to give a deterministic 2-approximation algorithm.

## 4.2.2 Comparison Sort Algorithms

We now describe three comparison sort algorithms. We have a relation  $\leq$  on the elements, where  $i \leq j$  if the majority of the input rankings has ranked  $i$  before  $j$  (in other words, if  $w_{(i,j)} \geq w_{(j,i)}$ ). We will say that  $i < j$  if  $i \leq j$  and  $j \not\leq i$ . We assume that the numbering of the elements is used to break ties, so that if both  $i \leq j$  and  $j \leq i$ , and  $i < j$  then we say that  $i < j$ .

Note that the relation  $<$  is not transitive, which is exactly Condorcet's paradox. The fact that the relation is not transitive implies that different comparison sort algorithms can produce different rankings.

### QuickSort

The QuickSort algorithm recursively sorts the elements by choosing a vertex  $i$  as pivot, and ordering vertex  $j$  to the left of (higher than)  $i$  if  $j < i$ , or to the right of  $i$  if  $i < j$  (breaking ties arbitrarily). The algorithm then recurses on the instance induced by the vertices to the left of  $i$  and those to the right of  $i$ .

Ailon et al. [4] showed that if we define  $\leq$  as above, and choose a pivot uniformly

at random, then the permutation returned by the QuickSort algorithm is an expected 2-approximation algorithm.

In the previous chapter, we gave a deterministic QuickSort algorithm, in which the “best” pivot is chosen instead of a random pivot, and we showed that this is also a 2-approximation algorithm. In particular, we compute the ratio in (3.1) for every vertex  $k$  and the element for which the ratio is smallest is chosen as pivot.

Compared to Ailon et al.’s QuickSort algorithm, the running time is approximately a factor of  $n$  slower, because in each iteration every potential pivot is evaluated. We therefore also consider a hybrid of the randomized and deterministic QuickSort algorithm, in which in a recursive call on  $V'$  we compute the ratio in (3.1) for  $\log(|V'|)$ , or a constant number of randomly chosen elements in  $V'$ , and choose the element that has the smallest ratio among these. In Section 4.4 we denote by QuickSort the original randomized algorithm, DetQuickSort the fully derandomized algorithm, LogQuickSort the algorithm which takes the best among  $\log(|V'|)$  pivots, and ConstQuickSort $k$ , which takes the best among  $k$  pivots.

## MergeSort

MergeSort recursively sorts the elements by dividing them into two (approximately) equal parts, recursing on each part to obtain two sorted lists, and merging the two lists as follows. We refer to the two sorted lists as List 1 and List 2, and we construct a merged list, List 3. While List 1 and List 2 are not empty, let  $i$  be the top element of List 1 and  $j$  the top element of List 2. If  $i < j$ , then we remove  $i$  from List 2 and add it to the bottom of List 3. Otherwise we move  $j$  to the bottom of List 3. Once one of List 1 and List 2 is empty, we add the remainder of the other list to the bottom of List 3.



## InsertionSort

In the InsertionSort algorithm, we start with an empty list and add the elements one by one to the list. When an element  $i$  is added to the list, it is placed in the highest position so that  $i < j$  for all elements  $j$  that are in lower positions than  $i$ . We can find  $i$ 's position by adding  $i$  to the bottom of the list, and allowing  $i$  to “bubble up”: while  $i < j$  for the element  $j$  directly above  $i$  in the list, we swap  $i$  and  $j$ . We note that the Local Kemenization procedure proposed in Dwork et al. [19] is the same as InsertionSort.

### 4.2.3 Existing Hybrid Algorithms

We now discuss algorithms that we call “hybrid” algorithms: they combine properties of comparison sort and positional algorithms. We first discuss known hybrid algorithms, and we then propose ways of combining the three comparison sort algorithms with any non-comparison sort method to obtain new hybrid algorithms.

#### Copeland's Method

We define the Copeland score of an element  $i$  to be the number of elements  $j$  such that  $i < j$ . Copeland [15] suggested sorting the elements by their Copeland score.

Note that if we define the majority tournament  $G = (V, A)$  to be the directed graph that has a node for every element, and an arc from  $i$  to  $j$  if  $i < j$ , then Copeland's method sorts the elements by non-increasing indegree. Hence Copeland's method can be seen as Borda's method on the majority tournament.

## MC4

Dwork et al. [19] propose Markov chain algorithms for rank aggregation that are similar to PageRank. We consider here the best of this type of algorithms that were considered by Dwork et al.: the MC4 algorithm. One way to think of this algorithm is to think of a random process on the set of elements. The process starts at some element  $i$ , and chooses one of the elements, say  $j$ , uniformly at random. If a majority of the input rankings prefers  $j$  to  $i$ , then we move to  $j$ , otherwise we stay at  $i$ . We then again choose an element at random and move if this element is preferred to the current element by a majority of the input rankings, etc.

This is known as a Markov process, where the transition matrix  $P$  has  $P(i, j) = \frac{1}{n}$  if a majority of the input rankings prefer  $j$  to  $i$ , and  $P(i, i) = 1 - \sum_{j \neq i} P(i, j)$ . Under certain conditions, this process has a unique (up to scalar multiples) limiting distribution  $x$  that satisfies  $x = xP$ , where  $x(i)$  gives the fraction of time the process spends at element  $i$ . Dwork et al. propose sorting the elements by non-increasing  $x(i)$  values.

To ensure that the process has a unique limiting distribution  $x$ , we use a “random jump”: with probability  $\delta > 0$ , we will choose a random element and move to this element (regardless of whether this element is preferred to the current element). In our experiments we have used  $\delta = \frac{1}{7}$ , which is the value of  $\delta$  that is often chosen in the literature for PageRank implementations.

In addition to calculating  $x$  exactly, we also consider the method MC4Approx, in which we start with a random vector  $y$ , and sort the elements according to  $\hat{x} = yP^n$ .

As Dwork et al. point out, the MC4 algorithm is similar in flavor to Copeland’s method: the diagonal entries of the transition matrix are exactly  $\frac{1}{n}$  times the Copeland scores of the elements.

#### 4.2.4 New Hybrids: Positional plus Comparison Sort Algorithms

We want to further exploit the fact that the rank aggregation problem has these two different approaches by combining them into new algorithms. We note that, except for the deterministic QuickSort algorithm of the previous chapter, the comparison sort algorithms make random choices, either by choosing an element to pivot on, by choosing how to divide the elements into two groups, or by choosing the order in which to insert the elements. One way to make these choices deterministic is to give a permutation as additional input to the comparison sort algorithms, and let the previously random choice be determined by the permutation instead. Hence we can use the permutation that is output by one algorithm as additional input to a comparison sort algorithm, thus obtaining a deterministic algorithm that hopefully combines the desirable qualities of the individual algorithms.

In the case of InsertionSort, it is clear how a permutation dictates the random choices made by the algorithm, since we can think of the permutation as giving the order in which to insert the elements. We also tested InsertionSort when the elements are inserted in the reverse order of the permutation. We note that the first approach was also suggested by Dwork et al. [19]. We will denote the first approach by IS, and the second by IS2. The results of IS2 are omitted, because IS tended to outperform IS2.

In MergeSort, the algorithm repeatedly divides the elements into two approximately equal parts and recurses on each part. We use a permutation to guide how the algorithm divides the elements. We tried two different approaches here: in MS we divide the elements according to whether they are in odd positions or in even positions. In MS2 we divide the elements according to whether their position is before or after the median position of the elements in the recursive call. Because MS performed better on most instances, we omitted the results of MS2 from our results.

Finally, we can use an input permutation to determine which element to pivot on in QuickSort. In the algorithm QS we take as pivot the element that is in the median position among the elements in the recursive call.

We will use the outputs of each of the positional methods as well as the output of the Copeland and MC4 algorithm as input into the different comparison sort methods.

## 4.2.5 Other Approaches

### Linear Programming

We solve the following linear programming relaxation, which we also saw in Section 3.2.3, to give a lower bound on the optimal value of a rank aggregation instance.

$$\begin{aligned}
\min \quad & \sum_{i,j \in V} w_{(i,j)} x_{(j,i)} + w_{(j,i)} x_{(i,j)} \\
\text{s.t.} \quad & x_{(i,j)} + x_{(j,k)} + x_{(k,i)} \geq 1 \quad \forall i, j, k \in V \\
& x_{(i,j)} + x_{(j,i)} = 1 \quad \forall i, j \in V \\
& x_{(i,j)} \geq 0.
\end{aligned}$$

In addition to providing a lower bound on the optimal value, we could use a comparison sort algorithm to round the fractional solution, where we now define  $i \leq j$  if  $x_{(i,j)} \geq x_{(j,i)}$ . We showed in Section 3.5 that this improves the performance guarantee of the QuickSort algorithm. However, we have not implemented this, because, surprisingly, the optimal solution to the linear program was integral on almost all our instances.

To speed up the solution of the linear program, we partitioned the instances into subinstances: It is not hard to show that if we can partition  $V$  into  $A, B$  such that  $w_{(i,j)} \geq w_{(j,i)}$  for all  $i \in A, j \in B$ , then there exists an optimal solution to the linear program that

has  $x_{(i,j)} = 1$  for all  $i \in A, j \in B$ . We used this fact to break up the LP for an instance into several LPs that could be solved separately.

Little is known about the integrality gap of this linear program, if the weights  $w$  arise from a rank aggregation instance. On our data sets, most instances had integer optimal solutions, and otherwise the gap between the optimal integer and optimal fractional solution was less than 0.002%.

## Local Search

We consider single vertex moves as a local search subroutine, which we apply as a clean-up step after each algorithm. Given a permutation  $\pi$ , a single vertex move takes an element  $i$  and inserts it into another position if this improves the objective value.

To the authors' knowledge there is no known performance guarantee for a permutation that is locally optimal with respect to single vertex moves. The example given in Coleman and Wirth [16] in which the local optimum is a factor  $\Omega(n)$  more expensive than the global optimum applies only to the feedback arc set problem with probability constraints, and not to rank aggregation since the weights do not satisfy the triangle inequality. There is evidence that single vertex moves are very powerful: they are part of the PTAS for the weighted feedback arc set problem with probability constraints by Kenyon-Mathieu and Schudy [53], and have been shown to be successful in other implementation studies [16].

We also investigate the algorithm called Chanas in Coleman and Wirth [16], which was proposed by Chanas and Kobylański [12]. This algorithm is a composition of sort and reverse steps. We start with a random permutation, and repeatedly go through the elements from left to right; if we can improve the objective by moving an element to the

left, we do so. Once we cannot make any improvements, we reverse the permutation, and repeat. Chanas and Kobylański show that for any permutation  $\pi$ , and the permutation  $\pi'$  we obtain by reversing  $\pi$  and one sorting pass through the elements, that the objective of  $\pi'$  is not more than that of  $\pi$ .

### 4.3 Lower Bounds on Guarantees

In the discussion of the algorithms, we noted that some of them have known upper bounds on their performance guarantees. We now show examples in which the algorithms do not perform that well, thus giving lower bounds on their performance guarantees.

#### 4.3.1 Positional Methods

We describe an example that can be turned into a bad example for both the Borda and Footrule algorithms. We have  $n$  elements, and our input permutations consist of  $n$  different permutations of the elements,  $\pi_1, \dots, \pi_n$ , where  $\pi_\ell$  occurs  $\mu_\ell$  times. The permutations are defined as  $\text{list}(\pi_1) = (1, 2, \dots, n)$ ,  $\text{list}(\pi_2) = (n, 1, \dots, n-1)$ ,  $\text{list}(\pi_3) = (n-1, n, 1, \dots, n-2)$ , etc. up to  $\text{list}(\pi_n) = (2, \dots, n, 1)$ , or equivalently,  $\pi_\ell(i) = (i + \ell - 1) \bmod n$ .

Note that if  $\mu_\ell = 1$  for every  $\ell$ , then the average and median position of each element is the same. This means that the elements are indistinguishable for Borda and Footrule. It is not hard to show that in this case, both Borda and the Footrule method can return any permutation. We will use this example with a small twist to give lower bounds on the performance guarantees for the Borda and Footrule method.

**Lemma 54** *The performance guarantee of Borda's method is at least 2.*

**Proof:** Let  $m$  be an arbitrary nonnegative constant. We let  $\mu_1 = m, \mu_n = m + n$  and  $\mu_\ell = m + n - 1$  for  $\ell = 2, \dots, n - 1$ .

Consider  $\text{Borda}(i) = \sum_{\ell=1}^n \frac{\mu_\ell}{M} \pi_\ell(i)$ , the Borda count of element  $i$ . In ranking  $\ell$ , the element is in position  $(i + \ell - 1) \bmod n$ , hence ranking  $\ell$  contributes  $((i + \ell - 1) \bmod n) \times \frac{\mu_\ell}{M}$  to the indegree of element  $i$ . We therefore get that

$$M\text{Borda}(i) = im + \sum_{\ell=2}^{n-i+1} (i + \ell - 1)(m + n - 1) + \sum_{\ell=n-i+2}^{n-1} (i + \ell - 1 - n)(m + n - 1) + (i - 1)(m + n).$$

We thus find that

$$\begin{aligned} & M(\text{Borda}(i) - \text{Borda}(i + 1)) \\ &= (im - (i + 1)m) + \left( \sum_{\ell=2}^{n-i+1} (i + \ell - 1)(m + n - 1) - \sum_{\ell=2}^{n-i} (i + \ell)(m + n - 1) \right) \\ &+ \left( \sum_{\ell=n-i+2}^{n-1} (i + \ell - 1 - n)(m + n - 1) - \sum_{\ell=n-i+1}^{n-1} (i + \ell - n)(m + n - 1) \right) \\ &+ ((i - 1)(m + n) - i(m + n)) \\ &= (-m) + \left( \sum_{\ell=2}^{n-i} (-1)(m + n - 1) + n(m + n - 1) \right) + \\ &\quad \left( \sum_{\ell=n-i+2}^{n-1} (-1)(m + n - 1) - (m + n - 1) \right) - (m + n) \\ &= -m + (i + 1)(m + n - 1) - (i - 1)(m + n - 1) - (m + n) \\ &= n - 2 > 0. \end{aligned}$$

Hence we see that Borda's method will order the elements in reverse order.

We compare the cost of Borda's ranking to the cost of the identity. We take  $m = n^2$  so that  $\frac{\mu_\ell}{M} = \frac{1}{n} + O(\frac{1}{n^2})$ . In the Borda ranking every pair is reversed, hence the cost of the

identity plus the cost of the Borda ranking should be equal to:

$$\begin{aligned} \sum_{\ell=1}^n \frac{\mu_{\ell}}{M} \frac{n(n-1)}{2} &= \sum_{\ell=1}^{n-1} \left( \frac{1}{n} + O\left(\frac{1}{n^2}\right) \right) \frac{n(n-1)}{2} \\ &= \frac{n^2}{2} + O(n). \end{aligned} \quad (4.2)$$

We now find the cost of the identity. Note that in  $\pi_1$ , no pairs are out of order with respect to the identity. In  $\pi_2$ , all pairs involving element  $n$  are out of order. In  $\pi_3$ , all pairs  $\{i, j\}$  such that  $i \in \{1, \dots, n-2\}, j \in \{n-1, n\}$ , are out of order. So in general, in the  $\ell$ -th ranking, we have  $(n-\ell+1)(\ell-1)$  pairs that are out of order. Hence the cost of the identity is

$$\begin{aligned} \sum_{\ell=1}^n \frac{\mu_{\ell}}{M} (n-\ell+1)(\ell-1) &= \sum_{\ell=0}^n \left( \frac{1}{n} + O\left(\frac{1}{n^2}\right) \right) (n-\ell)\ell \\ &= \frac{n^2}{6} + O(n). \end{aligned}$$

We subtract this from (4.2) to find that the objective value of the Borda ranking is  $\frac{n^2}{3} + O(n)$ , and hence the ratio of the cost for the Borda ranking compared to the identity tends to 2 if we let  $m = n$  and  $n \rightarrow \infty$ .  $\square$

**Lemma 55** *The performance guarantee of the Footrule method is at least 2.*

**Proof:** We again use the permutations  $\pi_1, \dots, \pi_n$ , where  $\pi_{\ell}(i) = (i + \ell - 1) \bmod n$ . We have one additional permutation  $\pi_{n+1}$  where  $\pi_{n+1}(i) = n - i + 1$ . We let  $\mu_{\ell} = m$  for  $\ell = 1, \dots, n$  and  $\mu_{n+1} = 1$ . The Footrule cost if we put element  $i$  in position  $\sigma(i)$  is  $\frac{1}{M} \sum_{\ell=1}^{n+1} \mu_{\ell} |\pi_{\ell}(i) - \sigma(i)| = \frac{m}{M} \sum_{\ell=1}^n |\pi_{\ell}(i) - \sigma(i)| + \frac{1}{M} |n - i + 1 - \sigma(i)|$ .

Note that for any  $i, j$  there is exactly one permutation in  $\pi_1, \dots, \pi_n$  such that  $\pi_{\ell}(i) = j$ . Hence  $\sum_{\ell=1}^n |\pi_{\ell}(i) - \sigma(i)| = \sum_{j=1}^n |j - \sigma(i)|$ . This expression does not depend on  $i$ , but only on  $\sigma(i)$ . Now, because  $\sigma$  is a permutation, for every  $k \in 1, \dots, n$  there is exactly one  $i$  such that  $\sigma(i) = k$ , therefore  $\sum_{i=1}^n \sum_{j=1}^n |j - \sigma(i)| = \sum_{k=1}^n \sum_{j=1}^n |j - k|$ .



Hence for any permutation  $\sigma$ , the average Footrule distance to the input rankings is

$$\frac{1}{M} \left( m \sum_{k=1}^n \sum_{j=1}^n |j - k| + |n - i + 1 - \sigma(i)| \right).$$

Hence the Footrule distance is minimized by setting  $\sigma(i) = n - i + 1$ .

We now let  $m = n$ , so that  $\frac{\mu_\ell}{M} = \frac{1}{n} + O(\frac{1}{n^2})$  for  $\ell = 1, \dots, n$  and  $\frac{\mu_{n+1}}{M} = 0 + O(\frac{1}{n^2})$ .

Then we know from the proof of Lemma 54 that the ratio of the objective value of  $\sigma$  compared to that of the identity tends to 2 if we let  $n \rightarrow \infty$ .  $\square$

The Pick-a-Perm algorithm does do well on the example given above; however, it is not hard to construct a bad example for Pick-a-Perm and Best-of- $k$ .

**Lemma 56** *The performance guarantee of Pick-a-Perm and Best-of- $k$  is at least 2.*

**Proof:** Let  $\pi_1, \dots, \pi_{n-1}$  be the input permutations, where  $\pi_i(j) = j$  if  $j \neq i, i + 1$  and  $\pi_i(i) = i + 1, \pi_i(i + 1) = i$ . Then the objective value of any of the input permutations is  $\frac{2(n-2)}{n-1}$ , but the objective value of the identity is 1.  $\square$

### 4.3.2 Comparison Sort Methods

With the exception of the deterministic QuickSort algorithm of the previous chapter, the comparison sort methods all need to make random choices: in (randomized) QuickSort the pivot is chosen uniformly at random from the elements, in MergeSort the elements are randomly divided into two equal sized groups, and in InsertionSort the elements are inserted in random order.

It is not difficult to devise examples together with a particular random choice for which these algorithms perform very badly, so that the algorithm's solution's objective

value could be a factor  $\Omega(n)$  higher than the optimal value. However, if with high probability the algorithm performs very well, one could just run the algorithm a few times, and take the best solution found.

In the case of InsertionSort, we show a much stronger result: there exists an example where, if inserting the elements in random order, the *expected* performance guarantee of the InsertionSort algorithm is  $\Omega(n)$ .

**Lemma 57** *The expected performance guarantee of InsertionSort is  $\Omega(n)$ .*

**Proof:** We consider an example with  $2n + 1$  elements, numbered 0 to  $2n$ . There are three input rankings, given by  $\pi_1, \pi_2, \pi_3$ , where  $\text{list}(\pi_1) = (0, 1, 2, \dots, n, n+1, \dots, 2n)$ ,  $\text{list}(\pi_2) = (1, \dots, n, n+1, \dots, 2n, 0)$  and  $\text{list}(\pi_3) = (n+1, \dots, 2n, 0, 1, \dots, n)$ , and  $\mu_1 = m, \mu_2 = m, \mu_3 = 1$  for some large constant  $m$ .

We call elements  $1, \dots, n$  red elements, and  $n+1, \dots, 2n$  blue elements. InsertionSort starts with an empty list, considers the element in random order and inserts the element in the highest position so that  $i < j$  for all elements  $j$  that are in lower positions than  $i$ .

Note that at the moment when 0 is considered, the current list has the elements considered thus far in lexicographical order. If some blue element has been considered before 0, then a blue element is at the bottom of the list, and 0 is inserted at the bottom. If a blue element is inserted next, it will be inserted in its correct position among the blue elements, but if a red element is inserted next, then it is inserted below 0. After the first red element that follows 0, all subsequent red and blue elements are inserted below 0.

We will let  $\mathcal{B}$  be blue elements that are considered before element 0, and we let  $\mathcal{R}$  be the red elements considered after element 0, and let  $B, R$  be the size of  $\mathcal{B}$  and  $\mathcal{R}$

respectively. From the previous discussion, InsertionSort ranks the elements in  $\mathcal{B}$  above the elements in  $\mathcal{R}$ , and the cost of the permutation returned by InsertionSort is thus at least  $\frac{2m}{2m+1}R \times B$ .

We note that if the elements are considered in random order then  $B$  and  $R$  are independent random variables, and  $B$  and  $n - B$ ,  $R$  and  $n - R$  are identically distributed, so the probability that both  $B$  and  $R$  are at least  $\frac{n}{2}$  is at least  $\frac{1}{4}$ . Hence  $\frac{2m}{2m+1}R \times B$  is at least  $\frac{2m}{2m+1} \frac{n^2}{4}$  with probability  $\frac{1}{4}$ , and the expectation of  $\frac{2m}{2m+1}R \times B$  is thus at least  $\frac{2m}{2m+1} \frac{n^2}{16}$ . On the other hand, the objective value for the permutations  $\pi_1, \pi_2$  is  $\frac{m+1}{2m+1}2n + \frac{1}{2m+1}n^2$ .  $\square$

Note that the example in the proof of Lemma 57 gives a bad example for QuickSort if we choose 0 as the first pivot, in which case the algorithm returns the solution  $(n + 1, \dots, 2n, 0, 1, \dots, n)$ , which is a factor  $\Omega(n)$  from optimal. However, one can show that the expected ratio of the objective value of the QuickSort solution and the optimal value for this particular example is not more than  $\frac{4}{3}$ . Similarly, if we use MergeSort on this example, and in the first recursive call, we split the elements into  $\{0, 1, \dots, n\}$  and  $\{n + 1, \dots, 2n\}$ , then MergeSort returns  $(n + 1, \dots, 2n, 0, 1, \dots, n)$  but the expected ratio of the MergeSort solution's objective value for this instance and the optimal value is constant.

### 4.3.3 Hybrid Algorithms

In the bad example for Borda's method in the previous section, both Copeland and the MC4 algorithm are not able to distinguish between the elements. We can adapt the example slightly, to make an example for which the MC4 algorithm returns a solution that costs a factor  $\frac{3}{2}$  more than the optimal value.

**Lemma 58** *The performance guarantee of the MC4 algorithm is at least  $\frac{3}{2}$ .*

**Proof:** We take  $n$  even, and we consider the same input rankings as in the bad example for the Borda and Footrule algorithms: We have  $n$  elements, and  $n$  permutations  $\pi_1, \dots, \pi_n$ , where  $\pi_\ell(i) = (i + \ell - 1) \bmod n$ , or, equivalently,  $\text{list}(\pi_\ell) = (n - \ell + 2, n - \ell + 3, \dots, n, 1, 2, \dots, n - \ell + 1)$ . The weights are given by  $\mu_1 = m, \mu_\ell = m + 1$  for  $2 \leq \ell \leq n$ .

Consider two elements  $i, j$ , where  $j > i$ . Then  $i$  is ranked before  $j$  in rankings 1 up to  $n - j + 1$ , and in rankings  $n - i + 2$  up to  $n$ . The number of voters that ranked  $i$  before  $j$  is thus  $m(n - (j - i)) + O(n)$  and the number of voters that ranked  $j$  before  $i$  is  $m(j - i) + O(n)$ . We choose  $m$  large enough so that the  $O(n)$  term is important only if  $n - (j - i) = j - i$ , i.e. if  $j - i = \frac{n}{2}$ . Then a majority of voters prefers  $j$  to  $i$  if  $j > i$  and  $j \geq i + \frac{n}{2}$ , or if  $j < i$  and  $i > j + \frac{n}{2}$ .

Therefore, we have

$$P(i, j) = \begin{cases} \frac{1}{n} & \text{if } i - \frac{n}{2} < j < i \text{ or } i + \frac{n}{2} \leq j \\ \frac{1}{2} & \text{if } i = j \leq \frac{n}{2} \\ \frac{1}{2} + \frac{1}{n} & \text{if } i = j > \frac{n}{2} \\ 0 & \text{otherwise.} \end{cases}$$

We show that the solution  $x$  to  $x = xP$  has  $x_{\frac{n}{2}+1} > x_{\frac{n}{2}+2} > \dots > x_n > x_{\frac{n}{2}} > x_{\frac{n}{2}-1} > \dots > x_1$ , and that the MC4 algorithm thus outputs the ranking  $\frac{n}{2} + 1, \frac{n}{2} + 2, \dots, n, \frac{n}{2}, \frac{n}{2} - 1, \dots, 1$ : Setting  $x = xP$  gives

$$\begin{aligned} x_i &= \frac{1}{2}x_i + \frac{1}{n}x_{i+1} + \dots + \frac{1}{n}x_{n/2-1+i}, \\ x_{n/2+i} &= \frac{1}{n}x_1 + \dots + \frac{1}{n}x_i + \left(\frac{1}{2} + \frac{1}{n}\right)x_{n/2+i} + \frac{1}{n}x_{n/2+i+1} + \dots + \frac{1}{n}x_n, \end{aligned}$$

for  $i = 1, 2, \dots, n/2$ .

Taking differences, we get:

$$\frac{1}{2}(x_i - x_{i+1}) = \frac{1}{n}(x_{i+1} - x_{n/2+i}), \quad (4.3)$$

$$\frac{1}{2}(x_{n/2} - x_{n/2+1}) = \frac{1}{n}(-x_1 - x_n), \quad (4.4)$$

$$\left(\frac{1}{2} - \frac{1}{n}\right)(x_{n/2+i} - x_{n/2+i+1}) = \frac{1}{n}(-x_{i+1} + x_{n/2+i+1}), \quad (4.5)$$

for  $i = 1, 2, \dots, n/2 - 1$ .

Adding (4.3) and (4.5) gives

$$\begin{aligned} \frac{1}{2}(x_i - x_{i+1}) &= \frac{1}{n}(x_{n/2+i+1} - x_{n/2+i}) - \left(\frac{1}{2} - \frac{1}{n}\right)(x_{n/2+i} - x_{n/2+i+1}) \\ &= \frac{1}{2}(x_{n/2+i+1} - x_{n/2+i}). \end{aligned} \quad (4.6)$$

**Claim 59**  $x_i < x_{i+1}$  implies  $x_{i+1} < x_{i+2}$  and  $x_{n/2+i} > x_{n/2+i+1}$  for  $i = 1, 2, \dots, n/2 - 2$ .

By contradiction. Assume  $x_i < x_{i+1}$  and  $x_{i+1} \geq x_{i+2}$ . The latter inequality together with (4.3) implies  $x_{i+2} \geq x_{n/2+i+1}$ . We thus have  $x_{i+1} \geq x_{i+2} \geq x_{n/2+i+1}$ , which together with (4.5) now implies  $x_{n/2+i} \leq x_{n/2+i+1}$  and hence  $x_{n/2+i} \leq x_{n/2+i+1} \leq x_{i+2} \leq x_{i+1}$ . However, the former inequality together with (4.3) implies  $x_{n/2+i} > x_{i+1}$ . The other part of the claim follows from (4.6).  $\diamond$

**Claim 60** *The equilibrium distribution  $x$  has  $x_1 < x_2 < \dots < x_{n/2} < x_n < x_{n-1} < \dots < x_{n/2+1}$ .*

(case 1) We will start by assuming that  $x_1 < x_2$ . We get  $x_1 < x_2 < \dots < x_{n/2}$  and  $x_n < x_{n-1} < \dots < x_{n/2+1}$ . Equation (4.5) with  $i = n/2 - 1$  gives us the remaining inequality  $x_{n/2} < x_n$ . Note that (4.4) indeed holds.

(case 2) Assume  $x_1 > x_2$ . Then we get in a similar way  $x_1 > x_2 > \dots > x_{n/2}$  and  $x_n > x_{n-1} > \dots > x_{n/2+1}$ . (4.3) with  $i = n/2 - 1$  gives the remaining inequality  $x_{n/2} > x_n$ . But now (4.4) is violated.

(case 3) Assume  $x_1 = x_2$ . Then we get in a similar way  $x_1 = x_2 = \dots = x_{n/2}$  and  $x_n = x_{n-1} = \dots = x_{n/2+1}$ . (4.3) with  $i = n/2 - 1$  gives  $x_{n/2} = x_n$ . (4.4) now tells us  $x_i = 0$  for all  $i$ , so this is not a distribution.  $\diamond$

We now compare the objective values of the MC4 solution and the identity. We take  $m = n$  so that  $\frac{\mu_\ell}{M} = \frac{1}{n} + O(\frac{1}{n^2})$  for  $\ell = 1, \dots, n$ .

The only pairs that are in the same order in the MC4 solution as in the identity are pairs  $\{i, j\}$  where  $i, j > \frac{n}{2}$ . We consider the cost for *these pairs* in the identity. In  $\pi_1$ , all these pairs are ordered lexicographically, in  $\pi_2$  the pairs with  $j = n$  are out of order, so there are  $\frac{n}{2} - 1$  out of order. In  $\pi_3$ , the pairs with  $\frac{n}{2} < i < n - 1$  and  $j \geq n - 1$  are out of order, and there are  $2(\frac{n}{2} - 2)$  of these. In general, in  $\pi_\ell$ , for  $\ell \leq \frac{n}{2}$ , there are  $(\ell - 1)(\frac{n}{2} - (\ell - 1))$  pairs  $\{i, j\}$  with  $i, j > \frac{n}{2}$  that are out of order with respect to the identity. For  $\ell > \frac{n}{2}$ , all pairs  $\{i, j\}$  with  $i, j > \frac{n}{2}$  are in lexicographical order in  $\pi_\ell$ . Hence the cost incurred by the identity (and hence also by the MC4 solution) for pairs  $\{i, j\}$  with  $i, j > \frac{n}{2}$  is

$$\begin{aligned} \sum_{\ell=1}^{n/2} \frac{\mu_\ell}{M} (\ell - 1) \left( \frac{n}{2} - (\ell - 1) \right) &= \sum_{\ell=0}^{n/2-1} \frac{\mu_\ell}{M} \ell \left( \frac{n}{2} - \ell \right) \\ &= \sum_{\ell=0}^{n/2-1} \left( \frac{1}{n} + O\left(\frac{1}{n^2}\right) \right) \ell \left( \frac{n}{2} - \ell \right) \\ &= \frac{1}{n} \frac{1}{6} \left( \frac{n}{2} \right)^3 + O(n) = \frac{n^2}{48} + O(n). \end{aligned}$$

From the proof of Lemma 54 we know that the total cost of the identity is  $\frac{n^2}{6} + O(n)$ . Hence the cost incurred by the identity *for all other pairs* (i.e. pairs where at least one of  $i, j$  is at most  $\frac{n}{2}$ ) is  $\frac{7n^2}{48} + O(n)$ . Now, note that there are  $(\frac{3}{8}n^2 - \frac{1}{4}n)$  pairs  $i, j$  such that at least

one of  $i, j$  is at most  $\frac{n}{2}$ . Since these pairs are in opposite order in the identity and the MC4 solution, the sum of the cost incurred by the two solutions for these pairs is  $(\frac{3}{8}n^2 - \frac{1}{4}n)$ . Therefore the cost for these pairs must be  $(\frac{3}{8}n^2 - \frac{1}{4}n) - (\frac{7n^2}{48} + O(n)) = \frac{11n^2}{48} + O(n)$  in the MC4 solution, and the total cost of the MC4 solution is thus  $\frac{n^2}{48} + \frac{11n^2}{48} + O(n) = \frac{n^2}{4} + O(n)$ . Since the objective value of the identity is  $\frac{n^2}{6} + O(n)$ , we get the result by letting  $n \rightarrow \infty$ .

□

Although the fact that MC4 performs better than Borda's method and Footrule aggregation on difficult examples like the ones we investigated is encouraging, to the best of the author's knowledge there is no performance guarantee known for the MC4 method.

We conclude this section by noting that none of the bad examples we considered is simultaneously a bad example for a positional method and for a comparison sort method. One can verify that executing any of the three comparison sort algorithms on the output of Borda's method or Footrule aggregation for their respective bad examples will return one of the input permutations, which all have objective value very close to optimal for  $m = n, n \rightarrow \infty$ . In the case of the bad example for Pick-a-Perm, the comparison function  $<$  is transitive, hence the identity is returned by any comparison sort algorithm. On the other hand, our bad example for InsertionSort is not a bad example for the positional methods. We do note however, that our new hybrid method BordaQS performs very badly on this example: the result of Borda's method has element 0 in the median position, and hence QS will pivot on this element first, thus returning the solution  $(n + 1, n + 2, \dots, 2n, 0, 1, \dots, n)$  which is a factor  $\Omega(n)$  more expensive than the optimal solution. This suggests that it is necessary for our new hybrid algorithms to output the better of the solution by the initial positional algorithm and the solution of subsequent hybrid algorithm.

## **4.4 Evaluation**

### **4.4.1 Description of Data Sets**

#### **Web Search Data Set**

We extracted search results from Ask, Google, MSN Live Search and Yahoo! using the default settings of each of these search engines. The queries we used for our experiment are the same 37 queries that were used by Dwork et al. [19]:

affirmative action, alcoholism, amusement parks, architecture, bicycling, blues, cheese, citrus groves, classical guitar, computer vision, cruises, Death Valley, field hockey, gardening, graphic design, Gulf war, HIV, java, Lipari, lyme disease, mutual funds, National parks, parallel architecture, Penelope Fitzgerald, recycling cans, rock climbing, San Francisco, Shakespeare, stamp collecting, sushi, table tennis, telecommuting, Thailand tourism, vintage cars, volcano, zen buddhism, and Zener.

As in the experiments of Dwork et al. we say that two pages are identical if their URLs are identical (up to some canonical form); we do not use the content of page to determine if two results are identical. We extracted the top-100 results from each search engine. On average, a single query resulted in 283 different pages. We assumed that all pages for a query that are returned by some search engine, but that are not in the top-100 of a particular search engine, are ranked at position 101 by that particular engine. The average number of results per query was 283, with a standard deviation of 23.4.



## Web Communities Data Set

We also used the Web Communities data set that was used by Coleman and Wirth [16] in their implementation study. We were not able to obtain the input rankings, but only the matrix  $w$  where  $w_{(i,j)}$  is the fraction of the input rankings that prefer  $i$  to  $j$ . For this reason, the only positional method we could evaluate on this data set is Borda’s method.

The Web Communities data set was obtained from 9 full rankings of 25 million documents [59]. From this data, 50 different inputs to the full rank aggregation problem were generated by choosing 50 samples of 100 documents each, and letting the input rankings be the induced 9 rankings on the 100 documents.

### 4.4.2 Results

We implemented each of the algorithms, except for the linear program and Footrule aggregation, in MATLAB. We found the optimal solutions to the linear program and Footrule aggregation using AMPL with the CPLEX solver.

The average solution time to find the LP optimum was 10.6 seconds for the Web Search data set, and 2.8 seconds for the Web Communities data set. We know from Section 3.5 that  $(LP_{FAS})$  has a small integrality gap: the integrality gap is at most  $\frac{3}{2}$  if the weights satisfy the triangle inequality, and at most  $\frac{4}{3}$  in the case of weights that arise from a full rank aggregation instance. In fact, in our data sets, the LP relaxation had an integer optimum for all instances in the Web Communities data set, and for all but 3 instances (corresponding to the queries “amusement parks”, “mutual funds” and “Shakespeare”) in the Web Search data set. The largest integrality gap was only .002%. The average time it took CPLEX to find the (integer) optimal solution is 3.5

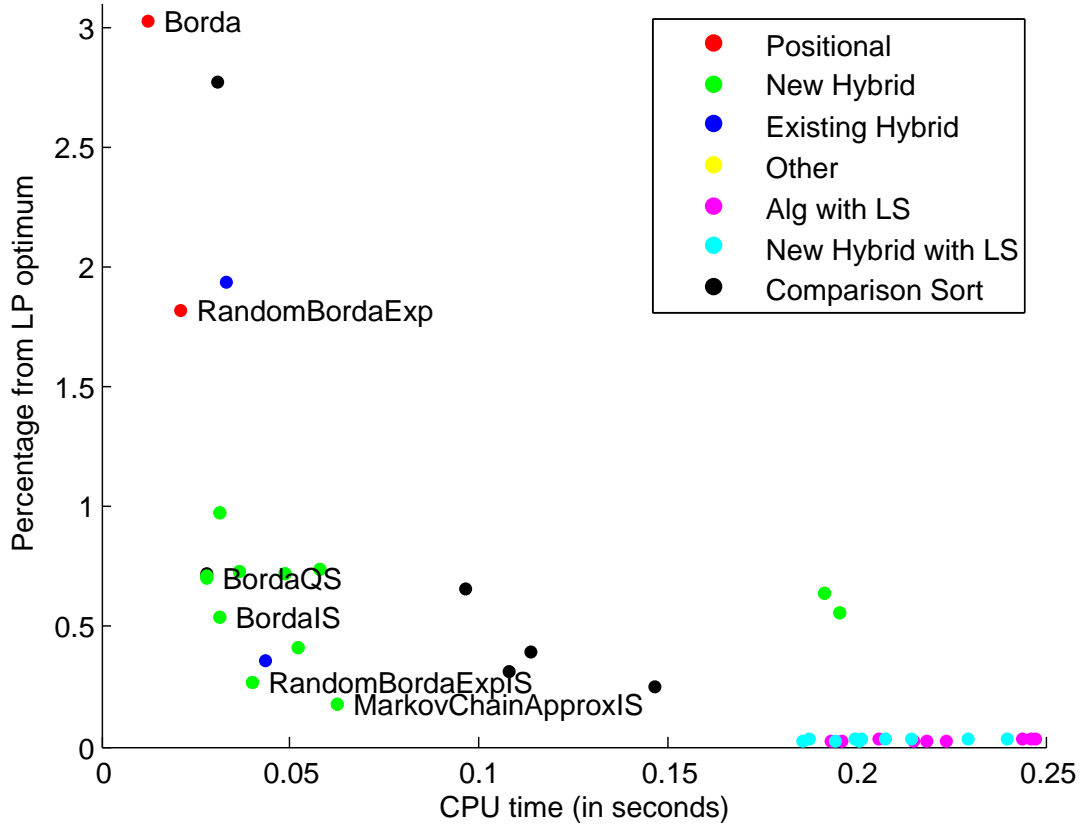


Figure 4.1: CPU time versus performance on the web search data. The name of an algorithm appears in the graph, if no algorithm with smaller running time performs better.

seconds for the Web Communities data set, and 132 seconds for the Web Search data set. The high average solution time for the Web Search data set is caused by one instance (corresponding to the query “Shakespeare”) for which it took CPLEX over an hour to find the optimal integer solution. The average solution time for the other instances in the Web Search data set was 4.6 seconds.

However, as we will see, the heuristics we study here find solutions solutions that cost less than 0.5% more than optimal in just a fraction of the time needed to solve the linear program.

For the randomized algorithms, we took the average objective value over 100 runs.

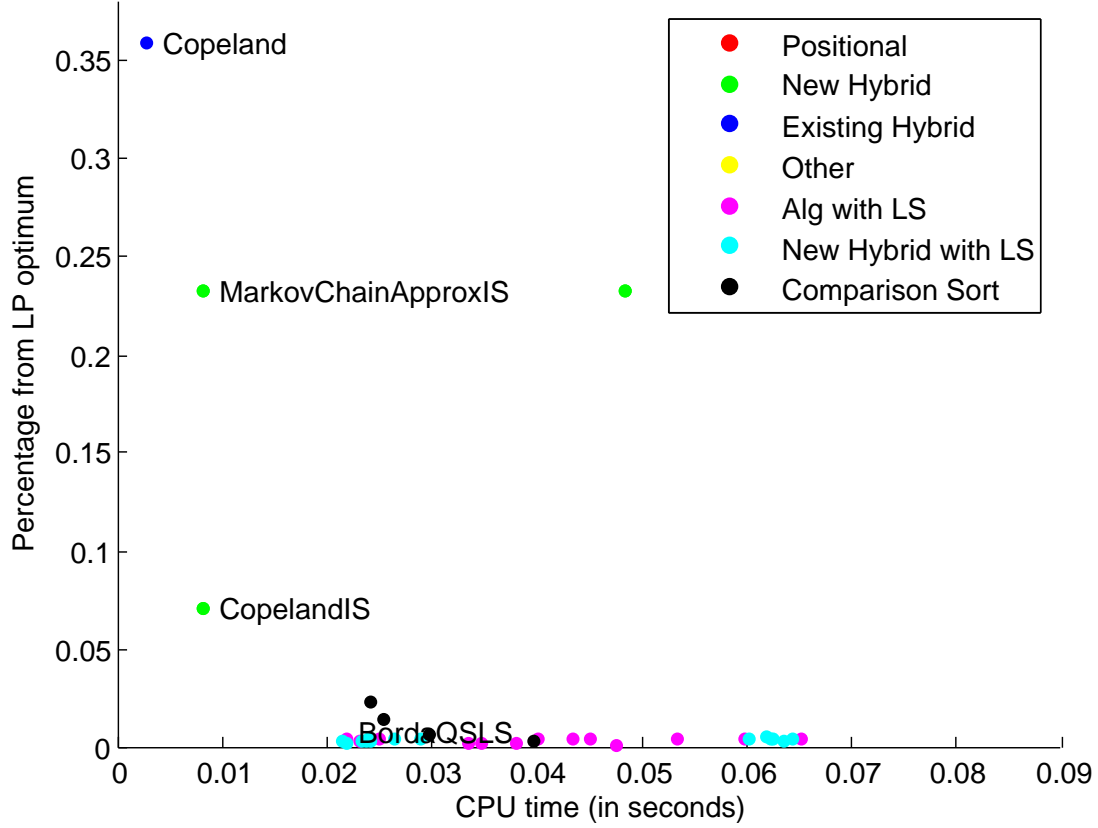


Figure 4.2: CPU time versus performance on web communities data. The name of an algorithm appears in the graph, if no algorithm with smaller running time performs better.

For each algorithm, and for each instance, we computed the percentage by which the algorithm’s solution value was higher than the LP optimum (we will refer to this as the “gap” of the algorithm). In the Appendix, we give four tables, two for each data set. For each data set, the first table contains the results without the local search clean-up step, and the second table gives the results after the local search clean-up. Each table shows the average CPU time and average gap. For comparison, we also included these values for a randomly generated permutation. In addition, we show the standard deviation of the gap over the different instances in the data set, and the (average) standard deviation of the gap for the randomized algorithms over the 100 random trials.

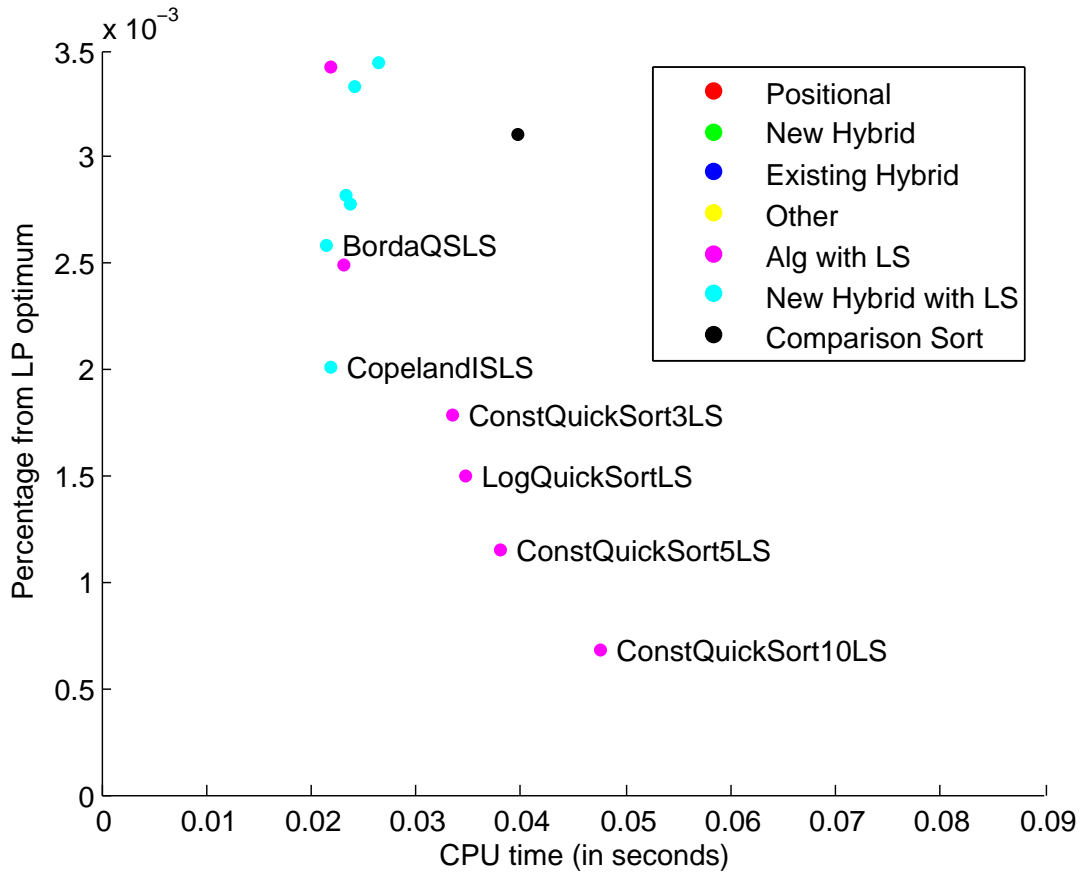


Figure 4.3: CPU time versus performance on web communities data, zoomed in on the right bottom corner of Figure 4.3 to show more detail.

For both data sets, the local search clean-up resulted in solutions extremely close to optimal; in the case of the Web Search data set, the remaining gap was between 0.02% and 0.04% and for the Web Communities data set, the remaining gap was between 0.001% and 0.005%. For the Web Search data set, the local search clean-up was relatively slow, taking on average 0.19 seconds. Surprisingly, on the Web Communities data set, the local search clean-up took only 0.03 seconds on average.

In Figures 4.1, 4.2 and 4.3 we display the CPU time versus the percentage by which the solution value found was higher than the LP optimum for all the algorithms we considered (including the combinations of algorithm plus local search clean-up, which are denoted with an appended “LS”). The name of an algorithm appears in the graph if

no algorithm with smaller running time performs better. Hence these graphs allow one to read off the best algorithm for a given computational budget.

The Web Communities data set seems to give a relatively easy set of instances, on which every hybrid algorithm, but also every comparison sort algorithm except for InsertionSort, performs very well. On the Web Search data set, the algorithms that use MergeSort do not perform well; in fact using MergeSort after a positional algorithm often makes the solution worse. On both data sets, QuickSort gives a very good randomized algorithm, or an excellent, but slow, deterministic algorithm. The positional methods are fast, and reasonably good on both data sets, but the hybrids need only a little bit of extra computation time, and perform much better.

#### **4.4.3 Randomized versus Deterministic QuickSort**

Because QuickSort also performs well on its own, not only in the hybrid algorithms, and to compare our deterministic QuickSort algorithm from Chapter 3 to the randomized QuickSort algorithm of Ailon et al. [4], we compared the different versions of QuickSort against each other if we allow each QuickSort algorithm the same amount of CPU time, where a randomized algorithm can use this CPU time to execute the algorithm multiple times, and return the best result found. We included the QuickSort algorithms that take the “best” of 3, 5, 10 or  $\log(|V'|)$  randomly chosen pivots, where “best” is in terms of the ratio in (3.1), in our comparison.

Based on the outcomes of the runs of the randomized algorithms, we computed the empirical probability distribution  $F$  of the “gap”, the percentage by which the algorithm’s solution value was higher than the LP optimum:  $F(x)$  is the fraction of the solutions for which the gap of the solution is at most  $x$ . We used 600 runs to obtain  $F$ .

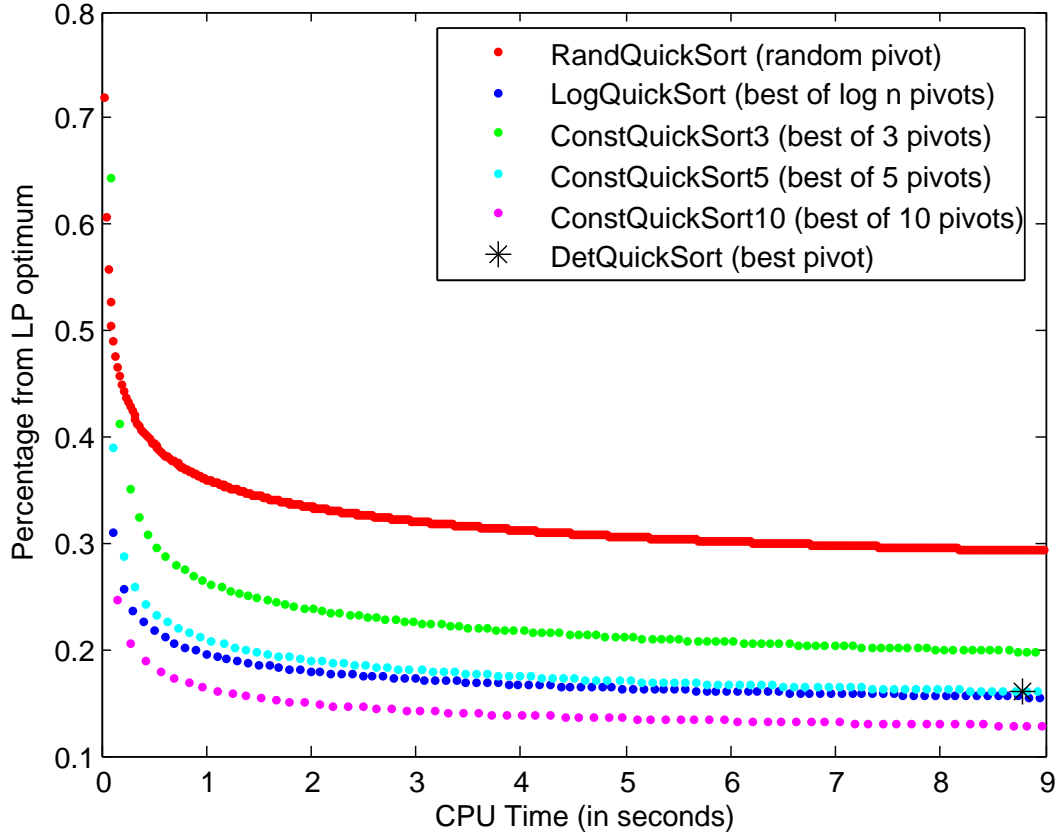


Figure 4.4: Best result after repeated runs of the QuickSort algorithms on Web Search Data.

Since the runs are independent, the probability that the best solution after  $k$  runs has a gap greater than  $x$  is approximately  $(1 - F(x))^k$ . Hence we have an approximation of the probability distribution of the best result after  $k$  runs. We computed the mean of this distribution for different values of  $k$  such that the computation time for  $k$  runs is not more than the computation time of the deterministic QuickSort algorithm. This way we obtain an estimate of the best result after  $k$  runs of the randomized algorithm on a particular instance. We plotted the mean of these estimates over the different instances against the mean time needed to run the randomized algorithm  $k$  times. The results are given in Figure 4.4 and 4.5. The  $x$ -axis gives the CPU time in seconds (for the appropriate number of runs) and the  $y$ -axis gives the mean gap for the best result after that number of

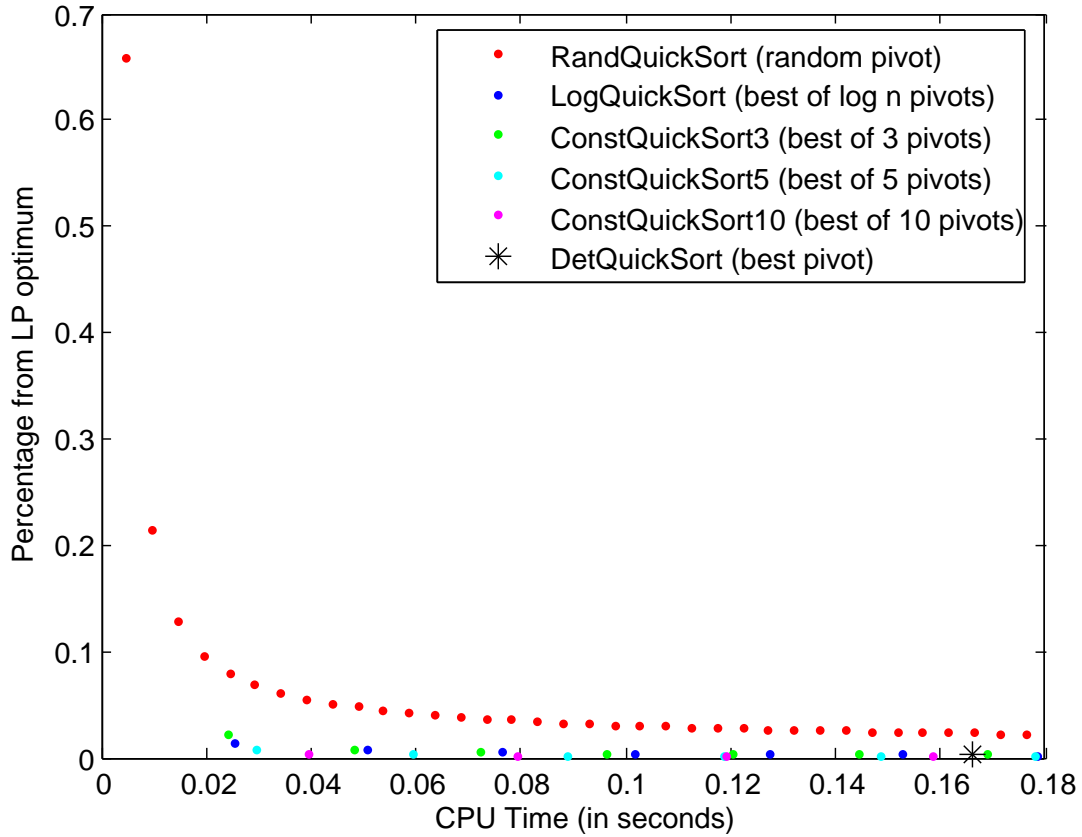


Figure 4.5: Best result after repeated runs of the QuickSort algorithms on Web Communities Data.

runs. In Figure 4.6 we show 90%-confidence intervals for the best result after repeated trials on the Web Search data set.

Although the deterministic QuickSort algorithm is too slow to be practical, the advantage of our analysis in Chapter 3 is very clear as well: on both data sets the randomized algorithms that evaluate a small number of randomly chosen potential pivots, and choose the one that minimizes (3.1) perform much better than the original randomized QuickSort algorithm, for any given computational budget.

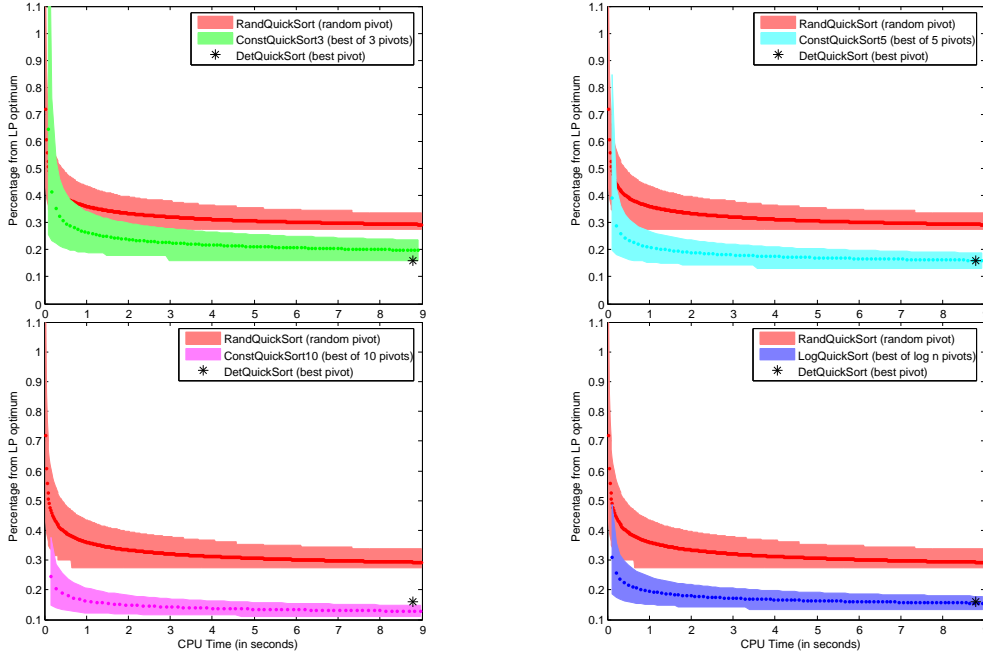


Figure 4.6: 90% Confidence Intervals for Best Result after Repeated Runs of the QuickSort Algorithms on Web Search Data.

## 4.5 Conclusion

We considered positional, comparison sort and algorithms that are hybrids of these two. There is some theoretical indication that this would yield improved algorithms, and we find in our evaluation that existing and our new hybrid methods give deterministic algorithms with an excellent trade-off of CPU time and performance, with the notable exception of the hybrids which involve MergeSort. The running time of the hybrid methods is not much higher than the running time of the positional methods, making them the best choice for fast deterministic algorithms. If speed is somewhat less of an issue, then the local search clean-up seems to give near-optimal results after almost any algorithm.

It is interesting that the local search algorithm performs especially well on the Web



Communities data, where each of the input rankings is a full ranking. It remains an open problem whether the local search algorithm is a constant factor approximation algorithm in the case of full rank aggregation. There is also nothing known about the theoretical performance guarantee of the MergeSort algorithm, although it seems quite clear from our experiments that this algorithm does not work well in practice. For InsertionSort on the other hand we were able to show that the performance guarantee is  $\Omega(n)$  if starting with a random input permutation, but the algorithm nonetheless performs very well on our data sets if starting with a good permutation.

Finally, in our experimental results we found that, although the deterministic algorithm we developed in Chapter 3 is too slow for practical purposes, our analysis can be used to obtain improved versions of the randomized QuickSort algorithm. These new randomized algorithms outperformed both the original randomized algorithm and our deterministic algorithm in running time and/or performance. On our data sets, the performance of these algorithms compares quite favorably to the other algorithms we considered.

# APPENDIX A

## TABLES WITH RESULTS FROM CHAPTER 4

Table A.1: The results of the algorithms on the web communities data set, without and with local search. “% gap” is the objective value of the algorithm’s solution minus the LP optimum divided by the LP optimum. Time is the CPU time in seconds. Std. 1 is the standard deviation of the algorithm’s % gap over the different instances. Std. 2 the (average) standard deviation of the % gap of the 100 random trials.

Algorithm	No Local Search				With Local Search			
	% gap	time	std. 1	std. 2	% gap	time	std. 1	std. 2
• randomPerm	188.74	0.0005	36.84	13.51	0.00	0.0535	0.00	0.01
• Borda	15.27	0.0005	1.78	0.00	0.00	0.0436	0.00	0.01
• RandBordaExp	17.73	0.0022	2.04	0.66	0.00	0.0468	0.00	0.01
• MergeSort	1.15	0.0069	0.59	1.97	0.00	0.0277	0.00	0.01
• InsertionSort	68.79	0.0056	24.21	25.51	0.00	0.0450	0.00	0.01
• QuickSort	0.66	0.0049	0.31	1.27	0.00	0.0219	0.00	0.01
• DetQuickSort	0.00	0.166	0.01	0.00	0.00	0.1737	0.00	0.00
• LogQuickSort	0.01	0.0255	0.01	0.02	0.00	0.0347	0.00	0.00
• ConstQuickSort3	0.02	0.0241	0.02	0.06	0.00	0.0335	0.00	0.00
• ConstQuickSort5	0.01	0.0297	0.01	0.01	0.00	0.0380	0.00	0.00
• ConstQuickSort10	0.00	0.0397	0.01	0.00	0.00	0.0476	0.00	0.00
• Copeland	0.36	0.0022	0.14	0.00	0.00	0.0251	0.00	0.00
• MC4	0.90	0.0297	0.43	0.00	0.00	0.0535	0.01	0.00
• MC4Approx	0.90	0.0025	0.43	0.01	0.00	0.0275	0.01	0.00
• BordaIS	0.63	0.0059	0.41	0.00	0.00	0.0265	0.01	0.00
• BordaMS	1.24	0.0071	2.91	0.00	0.00	0.0267	0.01	0.00
• BordaQS	0.99	0.0051	1.88	0.00	0.00	0.0214	0.01	0.00
• RandBordaExpIS	0.72	0.0079	0.42	0.21	0.00	0.0288	0.00	0.00
• RandBordaExpMS	1.51	0.0088	1.09	2.75	0.00	0.0290	0.00	0.01
• RandBordaExpQS	1.03	0.007	0.73	1.46	0.00	0.0238	0.00	0.00
• CopelandIS	0.07	0.0083	0.05	0.00	0.00	0.0218	0.01	0.00
• CopelandMS	0.75	0.0088	0.92	0.00	0.00	0.0291	0.01	0.00
• CopelandQS	0.43	0.0072	0.77	0.00	0.00	0.0241	0.01	0.00
• MC4IS	0.23	0.0484	0.22	0.00	0.00	0.0643	0.01	0.00
• MC4MS	0.95	0.0363	1.20	0.00	0.00	0.0570	0.01	0.00
• MC4QS	0.41	0.0474	1.13	0.00	0.00	0.0636	0.00	0.00
• MC4ApproxIS	0.23	0.0081	0.22	0.00	0.00	0.0240	0.01	0.00
• MC4ApproxMS	0.95	0.0091	1.20	0.01	0.00	0.0297	0.01	0.00
• MC4ApproxQS	0.41	0.0071	1.13	0.00	0.00	0.0234	0.00	0.00
• Chanas	0.00	0.5908	0.00	0.01	0.00	0.5978	0.00	0.01
• LP	0.00	2.8488	0.00	0.00	-	-	-	-

Table A.2: The results of the algorithms on the web search data set without and with local search. See Table A.1 for explanation of % gap, time, std. 1 and std. 2.

Algorithm	No Local Search				With Local Search			
	% gap	time	std. 1	std. 2	% gap	time	std. 1	std. 2
randomPerm	67.44	0.0119	13.87	4.62	0.03	0.2752	0.01	0.02
Pick-a-Perm	11.45	0.0125	2.49	4.20	0.03	0.2464	0.01	0.02
Best-of- $k$	5.88	0.1737	1.74	0.00	0.03	0.406	0.02	0.02
Borda	3.03	0.0126	0.90	0.00	0.02	0.1963	0.01	0.01
RandBordaExp	1.81	0.0207	0.23	0.09	0.02	0.2188	0.01	0.01
Footrule	11.25	3.148	1.41	0.00	0.03	3.3883	0.02	0.01
MergeSort	16.24	0.0478	0.57	2.16	0.03	0.2889	0.02	0.02
InsertionSort	2.77	0.0307	0.99	1.23	0.03	0.2151	0.01	0.02
QuickSort	0.72	0.0279	0.32	0.21	0.03	0.2057	0.02	0.02
DetQuickSort	0.16	8.7764	0.10	0.00	0.02	8.8916	0.02	0.01
LogQuickSort	0.31	0.1083	0.15	0.12	0.03	0.2439	0.02	0.02
ConstQuickSort3	0.65	0.0963	0.24	0.55	0.03	0.2472	0.02	0.02
ConstQuickSort5	0.39	0.1138	0.18	0.25	0.03	0.2531	0.02	0.02
ConstQuickSort10	0.25	0.1468	0.12	0.08	0.03	0.2737	0.02	0.02
Copeland	1.93	0.0333	0.41	0.00	0.03	0.2235	0.02	0.01
MC4	0.35	0.7195	0.13	0.00	0.02	0.8704	0.01	0.01
MC4Approx	0.35	0.0432	0.13	0.00	0.02	0.1935	0.01	0.01
Pick-a-PermIS	0.97	0.0315	0.46	0.44	0.03	0.2016	0.02	0.02
Pick-a-PermMS	10.44	0.0498	1.51	1.73	0.03	0.2911	0.02	0.02
Pick-a-PermQS	0.71	0.0279	0.31	0.19	0.03	0.2076	0.02	0.02
Best-of- $k$ IS	0.55	0.1953	0.30	0.00	0.03	0.3478	0.02	0.01
Best-of- $k$ MS	9.35	0.2139	2.38	0.00	0.03	0.447	0.02	0.01
Best-of- $k$ QS	0.64	0.1917	0.26	0.00	0.03	0.3703	0.03	0.01
BordaIS	0.53	0.0317	0.24	0.00	0.03	0.1877	0.02	0.01
BordaMS	15.27	0.0487	1.94	0.00	0.03	0.2878	0.02	0.01
BordaQS	0.70	0.0278	0.33	0.00	0.03	0.1998	0.02	0.01
RandBordaExpIS	0.26	0.0399	0.13	0.03	0.03	0.1855	0.02	0.01
RandBordaExpMS	15.85	0.0569	0.89	1.53	0.03	0.2983	0.01	0.02
RandBordaExpQS	0.73	0.0367	0.33	0.19	0.03	0.2145	0.02	0.02
FootruleIS	0.53	3.1669	0.25	0.00	0.03	3.3152	0.02	0.01
FootruleMS	16.66	3.1838	2.36	0.00	0.03	3.4239	0.02	0.01
FootruleQS	0.75	3.1641	0.37	0.00	0.03	3.3407	0.03	0.01
CopelandIS	0.42	0.0524	0.25	0.00	0.03	0.2008	0.02	0.01
CopelandMS	12.28	0.0707	2.43	0.00	0.03	0.3155	0.02	0.02
CopelandQS	0.72	0.0486	0.41	0.00	0.03	0.2293	0.02	0.01
MC4IS	0.18	0.7386	0.07	0.00	0.03	0.8708	0.02	0.01
MC4MS	5.94	0.7317	3.01	0.00	0.02	0.923	0.02	0.01
MC4QS	0.74	0.7344	0.37	0.00	0.03	0.9147	0.02	0.01
MC4ApproxIS	0.18	0.0623	0.07	0.00	0.03	0.1942	0.02	0.01
MC4ApproxMS	15.28	0.0794	1.59	0.00	0.02	0.3181	0.01	0.01
MC4ApproxQS	0.74	0.0581	0.37	0.00	0.03	0.2396	0.02	0.01
Chanas	0.02	2.92	0.01	0.01	0.02	2.9488	0.01	0.01
LP	0.00	10.6322	0.00	0.00	-	-	-	-

## BIBLIOGRAPHY

- [1] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Annals of Mathematics. Second Series*, 160(2):781–793, 2004.
- [2] Nir Ailon. Aggregation of partial rankings, p-ratings and top-m lists. In *Proceedings of the 18th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 415–424, 2007.
- [3] Nir Ailon and Moses Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 73–82, 2005.
- [4] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pages 684–693, 2005.
- [5] Noga Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, 20(1):137–142, 2006.
- [6] Sanjeev Arora, Alan M. Frieze, and Haim Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, pages 21–30, 1996.
- [7] Kenneth J. Arrow. *Social choice and individual values*. Yale University Press, 1951.
- [8] Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 21–29, 2001.
- [9] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
- [10] J.J. Bartholdi, C.A. Tovey, and M.A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.
- [11] J.C. Borda. *Memoire sur les elections au scrutin*. Histoire de l’Academie Royal des Sciences, 1781.

- [12] Stefan Chanas and Przemysław Kobylański. A new heuristic algorithm solving the linear ordering problem. *Computational Optimization and Applications*, 6(2):191–205, 1996.
- [13] Pierre Charbit, Stéphan Thomassé, and Anders Yeo. The minimum feedback arc set problem is NP-hard for tournaments. *Combinatorics, Probability and Computing*, 16(1):1–4, 2007.
- [14] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *Journal of Computer and System Sciences*, 71(3):360–383, 2005.
- [15] N. Christofides. Worst case analysis of a new heuristic for the traveling salesman problem. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.
- [16] Tom Coleman and Anthony Wirth. Ranking tournaments: Local search and a new algorithm. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*, 2008.
- [17] Don Coppersmith, Lisa Fleischer, and Atri Rudra. Ordering by weighted number of wins gives a good ranking for weighted tournaments. In *Proceedings of the 17th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 776–782, 2006.
- [18] Persi Diaconis and R. L. Graham. Spearman’s footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B. Methodological*, 39(2):262–268, 1977.
- [19] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the 10th International Conference on World Wide Web*, pages 613–622, 2001.
- [20] Friedrich Eisenbrand and Fabrizio Grandoni. An improved approximation algorithm for virtual private network design. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 928–932, 2005.
- [21] Friedrich Eisenbrand, Fabrizio Grandoni, Gianpaolo Oriolo, and Martin Skutella. New approaches for virtual private network design. *SIAM Journal on Computing*, 37(3):706–721, 2007.
- [22] Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Approximating connected facility location problems via random facility sampling

- and core detouring. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1174–1183, 2008.
- [23] Paul Erdős and Joel Spencer. *Probabilistic methods in combinatorics*. Academic Press, 1974.
  - [24] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.
  - [25] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing partial rankings. *SIAM Journal on Discrete Mathematics*, 20(3):628–648, 2006.
  - [26] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top  $k$  lists. *SIAM Journal on Discrete Mathematics*, 17(1):134–160, 2003.
  - [27] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 301–312, 2003.
  - [28] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *Journal of Computer and System Sciences*, 69(3):485–497, 2004.
  - [29] Vladimir Filkov and Steven Skiena. Integrating microarray data by consensus clustering. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pages 418–425, 2003.
  - [30] Lisa Fleischer, Jochen Könemann, Stefano Leonardi, and Guido Schäfer. Simple cost sharing schemes for multicommodity rent-or-buy and stochastic Steiner tree. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 663–670, 2006.
  - [31] Alan M. Frieze and Ravi Kannan. Quick approximation to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.
  - [32] Naveen Garg, Anupam Gupta, Stefano Leonardi, and Piotr Sankowski. Stochastic analyses for online combinatorial optimization problems. In *SODA*, pages 942–951, 2008.
  - [33] Naveen Garg, Rohit Khandekar, Goran Konjevod, R. Ravi, F. S. Salman, and

- Amitabh Sinha. On the integrality gap of a natural formulation of the single-sink buy-at-bulk network design problem. In *Proceedings of the 8th International Integer Programming and Combinatorial Optimization Conference*, pages 170–184, 2001.
- [34] Aristides Gionis, Heikki Mannila, Kai Puolamäki, and Antti Ukkonen. Algorithms for discovering bucket orders from data. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 561–566, 2006.
  - [35] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1):4, 2007.
  - [36] Andrey Goder and Vladimir Filkov. Consensus clustering algorithms: Comparison and refinement. In *Proceedings of the Workshop on Algorithm Engineering and Experiments*, 2008.
  - [37] Michel X. Goemans and Dimitris J. Bertsimas. Survivable networks, linear programming relaxations and the parsimonious property. *Mathematical Programming*, 60(2, Ser. A):145–166, 1993.
  - [38] Vineet Goyal, Anupam Gupta, Stefano Leonardi, and R. Ravi. Pricing tree access networks with connected backbones. In *Proceedings of the 15th Annual European Symposium on Algorithms*, pages 498–509, 2007.
  - [39] Fabrizio Grandoni and Giuseppe F. Italiano. Improved approximation for single-sink buy-at-bulk. In *Proceedings of the 17th International Symposium on Algorithms and Computation*, pages 111–120, 2006.
  - [40] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. A constant factor approximation for the single sink edge installation problems. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 383–388, 2001.
  - [41] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost-sharing: A simple approximation algorithm for the multicommodity rent-or-buy problem. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pages 606–615, 2003.
  - [42] Anupam Gupta, Amit Kumar, Martin Pál, and Tim Roughgarden. Approximation via cost sharing: simpler and better approximation algorithms for network design. *Journal of the ACM*, 54(3):11, 2007.

- [43] Anupam Gupta, Amit Kumar, and Tim Roughgarden. Simpler and better approximation algorithms for network design. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 365–372, 2003.
- [44] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: approximation algorithms for stochastic optimization. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 417–426, 2004.
- [45] Anupam Gupta, Martin Pál, Ramamoorthi Ravi, and Amitabh Sinha. What about Wednesday? Approximation algorithms for multistage stochastic optimization. In *Approximation, Randomization and Combinatorial Optimization*, volume 3624 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2005.
- [46] Anupam Gupta, Aravind Srinivasan, and Éva Tardos. Cost-sharing mechanisms for network design. *Algorithmica*, 50(1):98–119, 2008.
- [47] Boulos Harb, Sampath Kannan, and Andrew McGregor. Approximating the best-fit tree under  $L_p$  norms. In *Approximation, Randomization and Combinatorial Optimization*, volume 3624 of *Lecture Notes in Computer Science*, pages 123–133. Springer, Berlin, 2005.
- [48] Mohammad Khairul Hasan, Hyunwoo Jung, and Kyung-Yong Chwa. Approximation algorithms for connected facility location problems. *Journal of Combinatorial Optimization*, to appear, 2008.
- [49] Rajneesh Hegde and Kamal Jain. Personal communication.
- [50] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab S. Mirrokni. On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 691–700, 2004.
- [51] Richard M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34(1-3):165–201, 1991.
- [52] J.G. Kemeny. Mathematics without numbers. *Daedalus*, 88:575–591, 1959.
- [53] Claire Kenyon-Mathieu and Warren Schudy. How to rank with few errors. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 95–103, 2007.



- [54] Mirko Křivánek and Jaroslav Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.
- [55] Mohammad Mahdian, Yinyu Ye, and Jiawei Zhang. Approximation algorithms for metric facility location problems. *SIAM Journal on Computing*, 36(2):411–432 (electronic), 2006.
- [56] Dharmendra Modha and Scott Spangler. Feature weighting in k-means clustering. *Machine Learning*, 52(3):217–237, 2003.
- [57] Dharmendra S. Modha and W. Scott Spangler. Clustering hypertext with applications to web searching. In *Proceedings of the 11th ACM on Hypertext and Hypermedia*, pages 143–152, 2000.
- [58] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.
- [59] Laurence A. F. Park and Kotagiri Ramamohanarao. Mining web multi-resolution community-based popularity for information retrieval. In *Proceedings of the 16th ACM Conference on Information and Knowledge Management*, pages 545–552, November 2007.
- [60] Prabhakar Raghavan. Probabilistic construction of deterministic algorithms: approximating packing integer programs. *Journal of Computer and System Sciences*, 37(2):130–143, 1988.
- [61] Paul D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.
- [62] David Shmoys and Kunal Talwar. A constant approximation algorithm for the a priori traveling salesman problem. In *Proceedings of the 13th International Integer Programming and Combinatorial Optimization Conference*, 2008.
- [63] David B. Shmoys and David P. Williamson. Analyzing the Held-Karp TSP bound: A monotonicity property with application. *Information Processing Letters*, 35:281–285, 1990.
- [64] Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4), 2004.
- [65] Kunal Talwar. The single-sink buy-at-bulk LP has constant integrality gap. In *Pro-*

*ceedings of the 9th International Integer Programming and Combinatorial Optimization Conference*, pages 475–486, 2002.

- [66] Anke van Zuylen, Rajneesh Hegde, Kamal Jain, and David P. Williamson. Deterministic pivoting algorithms for constrained ranking and clustering problems. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 405–414, 2007.
- [67] Yoshiko Wakabayashi. The complexity of computing medians of relations. *Resenhas*, 3(3):323–349, 1998.
- [68] David P. Williamson and Anke van Zuylen. A simpler and better derandomization of an approximation algorithm for single source rent-or-buy. *Operations Research Letters*, 35(6):707–712, 2007.